



All-in at the River

Standard Code Library

Shanghai Jiao Tong University

Desprado2 fstqwq AntiLeaf

Contents

1 数学	1	
1.1 多项式	1	
1.1.1 FFT	1	
1.1.2 NTT	1	
1.1.3 任意模数卷积(MTT, 毛梯梯)	2	
1.1.4 多项式操作	3	
1.1.5 更优秀的多项式多点求值	5	
1.1.6 多项式快速插值	6	
1.1.7 拉格朗日反演(多项式复合逆)	8	
1.1.8 分治FFT	8	
1.1.9 半在线卷积	8	
1.1.10 常系数齐次线性递推 $O(k \log k \log n)$	8	
1.1.11 应用: $O(\sqrt{n} \log^2 n)$ 快速求阶乘	9	
1.2 插值	9	
1.2.1 牛顿插值	9	
1.2.2 拉格朗日(Lagrange)插值	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 Berlekamp-Massey最小递推式	11	
1.5.1 优化矩阵快速幂DP	12	
1.5.2 求矩阵最小多项式	12	
1.5.3 求稀疏矩阵的行列式	12	
1.5.4 求稀疏矩阵的秩	12	
1.5.5 解稀疏方程组	12	
1.6 单纯形	13	
1.6.1 线性规划对偶原理	13	
1.7 博弈论	14	
1.7.1 SG定理	14	
1.7.2 纳什均衡	14	
1.7.3 经典博弈	14	
1.7.4 例题	14	
1.8 自适应Simpson积分	14	
1.9 常见数列	14	
1.9.1 斐波那契数 卢卡斯数	14	
1.9.2 伯努利数, 自然数幂次和	15	
1.9.3 分拆数	15	
1.9.4 斯特林数	15	
1.9.5 贝尔数	16	
1.9.6 欧拉数(Eulerian Number)	16	
1.9.7 卡特兰数, 施罗德数, 默慈金数	16	
1.10 常用公式及结论	17	
1.10.1 方差	17	
1.10.2 min-max反演	17	
1.10.3 单位根反演(展开整除条件 $[n k]$)	17	
1.10.4 康托展开(排列的排名)	17	
1.10.5 连通图计数	17	
1.10.6 线性齐次线性常系数递推求通项	17	
1.11 常用生成函数变换	17	
2 数论	18	
2.1 $O(n)$ 预处理逆元	18	
2.2 线性筛	18	
2.3 杜教筛	18	
2.4 Powerful Number筛	18	
2.5 洲阁筛	19	
2.6 Miller-Rabin	20	
2.7 Pollard's Rho	20	
2.8 快速阶乘算法	21	
2.9 扩展欧几里德	21	
2.9.1 求通解的方法	21	
2.9.2 类欧几里德算法(直线下整点个数)	21	
2.10 中国剩余定理	21	
2.10.1 ex-CRT	21	
2.11 原根 阶	22	
2.12 常用数论公式	22	
2.12.1 莫比乌斯反演	22	
2.12.2 降幂公式	22	
2.12.3 其他常用公式	22	
3 图论	23	
3.1 最小生成树	23	
3.1.1 Boruvka算法	23	
3.1.2 动态最小生成树	23	
3.1.3 最小树形图	24	
3.1.4 Steiner Tree 斯坦纳树	25	
3.1.5 最小直径生成树	26	
3.2 最短路	27	
3.2.1 Dijkstra	27	
3.2.2 Johnson算法(负权图多源最短路)	27	
3.2.3 k 短路	27	
3.3 Tarjan算法	28	
3.3.1 强连通分量	28	
3.3.2 割点 点双	28	
3.3.3 桥 边双	29	
3.4 仙人掌	29	
3.4.1 仙人掌DP	29	
3.5 二分图	30	
3.5.1 匈牙利	30	
3.5.2 Hopcroft-Karp二分图匹配	30	
3.5.3 KM二分图最大权匹配	31	
3.5.4 二分图原理	32	
3.6 一般图匹配	32	
3.6.1 高斯消元	32	
3.6.2 带花树	33	
3.6.3 带权带花树	34	
3.6.4 原理	36	
3.7 支配树	36	
3.8 2-SAT	36	
3.9 最大流	37	
3.9.1 Dinic	37	
3.9.2 ISAP	37	
3.9.3 HLPP 最高标号预流推进	38	
3.10 费用流	39	
3.10.1 SPFA费用流	39	
3.10.2 Dijkstra费用流	40	
3.10.3 预流推进费用流(可处理负环) $O(nm \log C)$	41	
3.11 网络流原理	42	
3.11.1 最大流	42	
3.11.2 最小割	43	
3.11.3 上下界网络流	43	
3.11.4 常见建图方法	43	
3.11.5 例题	43	
3.12 Prufer序列	44	
3.13 弦图相关	44	
3.14 其他	44	
3.14.1 Stoer-Wagner全局最小割	44	
4 数据结构	46	
4.1 线段树	46	
4.1.1 非递归线段树	46	
4.1.2 线段树维护矩形并	46	

4.1.3 历史和	47	5.8 字符串原理	80
4.2 陈丹琦分治	48	6 动态规划	81
4.2.1 动态图连通性(分治并查集)	48	6.1 决策单调性 $O(n \log n)$	81
4.2.2 四维偏序	49	6.2 例题	81
4.3 整体二分	49	6.2.1 103388A Assigning Prizes 容斥	81
4.4 平衡树	50	7 计算几何	83
4.4.1 Treap	50	7.1 Delaunay三角剖分	83
4.4.2 无旋Treap/可持久化Treap	51	7.2 最近点对	85
4.4.3 Splay	52	8 杂项	86
4.5 树链剖分	52	8.1 $O(1)$ 快速乘	86
4.5.1 动态树形DP(最大权独立集)	52	8.2 Kahan求和算法(减少浮点数累加的误差)	86
4.6 树分治	54	8.3 Python Decimal	86
4.6.1 动态树分治	54	8.4 $O(n^2)$ 高精度	86
4.6.2 紫荆花之恋	55	8.5 笛卡尔树	89
4.7 LCT动态树	58	8.6 GarsiaWachs算法($O(n \log n)$ 合并石子)	89
4.7.1 不换根(弹飞绵羊)	58	8.7 常用NTT素数及原根	89
4.7.2 换根/维护生成树	58	8.8 xorshift	89
4.7.3 维护子树信息	60	8.9 枚举子集	90
4.7.4 模板题:动态QTREE4	61	8.10 STL	90
4.8 K-D树	63	8.10.1 vector	90
4.8.1 动态K-D树(定期重构)	63	8.10.2 list	90
4.9 LCA最近公共祖先	64	8.10.3 unordered_set/map	90
4.9.1 Tarjan LCA $O(n + m)$	64	8.11 Public Based DataStructure(PB_DS)	90
4.10 虚树	64	8.11.1 哈希表	90
4.11 长链剖分	66	8.11.2 堆	90
4.11.1 梯子剖分	66	8.11.3 平衡树	90
4.12 堆	67	8.12 rope	91
4.12.1 左偏树	67	8.13 其他C++相关	91
4.12.2 二叉堆	67	8.13.1 <cmath>	91
4.13 莫队	67	8.13.2 <algorithm>	91
4.13.1 莫队二次离线	67	8.13.3 std::tuple	91
4.13.2 带修莫队在线化 $O(n^{\frac{5}{3}})$	69	8.13.4 <complex>	91
4.13.3 莫队二次离线 在线化 $O((n + m)\sqrt{n})$	69	8.14 一些游戏	91
4.14 常见根号思路	70	8.14.1 德州扑克	91
5 字符串	71	8.14.2 炉石传说	94
5.1 KMP	71	8.15 OEIS	94
5.1.1 ex-KMP	71	8.15.1 计数相关	94
5.2 AC自动机	71	8.15.2 线性递推数列	95
5.3 后缀数组	72	8.15.3 数论相关	95
5.3.1 倍增	72	8.15.4 其他	95
5.3.2 SA-IS	72	8.16 编译选项	95
5.3.3 SAMSA	73	8.17 附录: VScode相关	95
5.4 后缀平衡树	74	8.17.1 插件	95
5.5 后缀自动机	74	8.17.2 设置选项	96
5.5.1 广义后缀自动机	74	8.17.3 快捷键	96
5.5.2 区间本质不同子串计数	75	8.18 附录: 骂人的艺术—梁实秋	96
5.6 回文树	78	8.19 附录: Cheat Sheet	96
5.6.1 广义回文树	78		
5.7 Manacher马拉车	80		

1 数学

1.1 多项式

1.1.1 FFT

```

1 // 使用时一定要注意double的精度是否足够(极限大概是10 ^ ↪ 14)
2
3 const double pi = acos((double)-1.);
4
5 // 手写复数类
6 // 支持加减乘三种运算
7 // += 运算符如果用的不多可以不重载
8 struct Complex {
9     double a, b; // 由于long double精度和double几乎相同,
    → 通常没有必要用long double
10
11     Complex(double a = 0., double b = 0.) : a(a), b(b)
    → {}
12
13     Complex operator + (const Complex &x) const {
14         return Complex(a + x.a, b + x.b);
15     }
16
17     Complex operator - (const Complex &x) const {
18         return Complex(a - x.a, b - x.b);
19     }
20
21     Complex operator * (const Complex &x) const {
22         return Complex(a * x.a - b * x.b, a * x.b + b * ↪ x.a);
23     }
24
25     Complex operator * (double x) const {
26         return Complex(a * x, b * x);
27     }
28
29     Complex &operator += (const Complex &x) {
30         return *this = *this + x;
31     }
32
33     Complex conj() const { // 共轭, 一般只有MTT需要用
34         return Complex(a, -b);
35     }
36 } omega[maxn], omega_inv[maxn];
37 const Complex ima = Complex(0, 1); // i = sqrt(-1)
38
39 int fft_n; // 要在主函数里初始化
40
41 // FFT初始化
42 void FFT_init(int n) {
43     fft_n = n;
44
45     for (int i = 0; i < n; i++) // 根据单位根的旋转性质
        → 可以节省计算单位根逆元的时间
        omega[i] = Complex(cos(2 * pi / n * i), sin(2 * ↪ pi / n * i));
46
47     omega_inv[0] = omega[0];
48     for (int i = 1; i < n; i++)
        omega_inv[i] = omega[n - i];
49
50     // 当然不存单位根也可以, 只不过在FFT次数较多时很可能
        → 会增大常数
51 }
52
53 // FFT主过程
54 void FFT(Complex *a, int n, int tp) {
55     for (int i = 1, j = 0, k; i < n - 1; i++) {

```

```

57         k = n;
58         do
59             j ≈ (k >= 1);
60         while (j < k);
61
62         if (i < j)
63             swap(a[i], a[j]);
64     }
65
66     for (int k = 2, m = fft_n / 2; k ≤ n; k *= 2, m /= ↪ 2)
67         for (int i = 0; i < n; i += k)
68             for (int j = 0; j < k / 2; j++) {
69                 Complex u = a[i + j], v = (tp > 0 ?
                    → omega : omega_inv)[m * j] * a[i + j
                    → + k / 2];
70
71                 a[i + j] = u + v;
72                 a[i + j + k / 2] = u - v;
73             }
74
75         if (tp < 0)
76             for (int i = 0; i < n; i++) {
77                 a[i].a /= n;
78                 a[i].b /= n; // 一般情况下是不需要的, 只
                    → 有MTT时才需要
79         }
80     }

```

1.1.2 NTT

```

1 constexpr int p = 998244353; // p为模数
2
3 int ntt_n, omega[maxn], omega_inv[maxn]; // ntt_n要在主
    → 函数里初始化
4
5 void NTT_init(int n) {
6     ntt_n = n;
7
8     int wn = qpow(3, (p - 1) / n); // 这里的3代表模数的
    → 任意一个原根
9
10    omega[0] = omega_inv[0] = 1;
11
12    for (int i = 1; i < n; i++)
13        omega_inv[n - i] = omega[i] = (long
            → long)omega[i - 1] * wn % p;
14
15
16 void NTT(int *a, int n, int tp) { // n为变换长度,
    → tp为1或-1, 表示正/逆变换
17
18     for (int i = 1, j = 0, k; i < n - 1; i++) { // → O(n)旋转算法, 原理是模拟加1
19         k = n;
20         do
21             j ≈ (k >= 1);
22         while (j < k);
23
24         if (i < j)
25             swap(a[i], a[j]);
26     }
27
28     for (int k = 2, m = ntt_n / 2; k ≤ n; k *= 2, m /= ↪ 2)
29         for (int i = 0; i < n; i += k)
30             for (int j = 0; j < k / 2; j++) {
31                 int w = (tp > 0 ? omega : omega_inv)[m
                    → * j];

```

```

32     int u = a[i + j], v = (long long)w *
33         ↪ a[i + j + k / 2] % p;
34     a[i + j] = u + v;
35     if (a[i + j] ≥ p)
36         a[i + j] -= p;
37
38     a[i + j + k / 2] = u - v;
39     if (a[i + j + k / 2] < 0)
40         a[i + j + k / 2] += p;
41 }
42
43 if (tp < 0) {
44     int inv = qpow(n, p - 2);
45     for (int i = 0; i < n; i++)
46         a[i] = (long long)a[i] * inv % p;
47 }

```

1.1.3 任意模数卷积(MTT, 毛梯梯)

三模数NTT和直接拆系数FFT都太慢了, 不要用.

MTT的原理就是拆系数FFT, 只不过优化了做变换的次数.

考虑要对 $A(x), B(x)$ 两个多项式做DFT, 可以构造两个复多项式

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

只需要DFT一个, 另一个DFT实际上就是前者反转再取共轭, 再利用

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

即可还原出 $A(x), B(x)$.

IDFT的道理更简单, 如果要对 $A(x)$ 和 $B(x)$ 做IDFT, 只需要对 $A(x) + iB(x)$ 做IDFT即可, 因为IDFT的结果必定为实数, 所以结果的实部和虚部就分别是 $A(x)$ 和 $B(x)$.

实际上任何同时对两个实序列进行DFT, 或者同时对结果为实序列的DFT进行逆变换时都可以按照上面的方法优化, 可以减少一半的DFT次数.

```

30     FFT(c, n, -1);
31
32     for (int i = 0; i < n; i++) {
33         a[i].a = c[i].a;
34         b[i].a = c[i].b;
35     }
36 }
37
38 Complex a[2][maxn], b[2][maxn], c[3][maxn];
39 int ans[maxn];
40
41 int main() {
42     int n, m;
43     scanf("%d%d%d", &n, &m, &p);
44     n++;
45     m++;
46
47     base = (int)(sqrt(p) + 0.5);
48
49     for (int i = 0; i < n; i++) {
50         int x;
51         scanf("%d", &x);
52         x %= p;
53
54         a[1][i].a = x / base;
55         a[0][i].a = x % base;
56     }
57
58     for (int i = 0; i < m; i++) {
59         int x;
60         scanf("%d", &x);
61         x %= p;
62
63         b[1][i].a = x / base;
64         b[0][i].a = x % base;
65     }
66
67     int N = 1;
68     while (N < n + m - 1)
69         N <<= 1;
70
71     FFT_init(N);
72
73     DFT(a[0], a[1], N);
74     DFT(b[0], b[1], N);
75
76     for (int i = 0; i < N; i++)
77         c[0][i] = a[0][i] * b[0][i];
78
79     for (int i = 0; i < N; i++)
80         c[1][i] = a[0][i] * b[1][i] + a[1][i] * b[0]
81             ↪ [i];
82
83     for (int i = 0; i < N; i++)
84         c[2][i] = a[1][i] * b[1][i];
85
86     FFT(c[1], N, -1);
87     IDFT(c[0], c[2], N);
88
89     for (int j = 2; ~j; j--)
90         for (int i = 0; i < n + m - 1; i++)
91             ans[i] = ((long long)ans[i] * base + (long
92                 ↪ long)(c[j][i].a + 0.5)) % p;
93
94     // 实际上就是c[2] * base ^ 2 + c[1] * base + c[0],
95     // 这样写可以改善地址访问连续性
96
97     for (int i = 0; i < n + m - 1; i++) {
98         if (i)
99
100            ans[i] = ((long long)ans[i] * base + (long
101                ↪ long)(c[2][i].a + 0.5)) % p;
102
103     }
104
105     printf("%d\n", ans[n + m - 1]);
106 }

```

```

1 // 常量和复数类略
2
3 const Complex ima = Complex(0, 1);
4
5 int p, base;
6
7 // FFT略
8
9 void DFT(Complex *a, Complex *b, int n) {
10     static Complex c[maxn];
11
12     for (int i = 0; i < n; i++)
13         c[i] = Complex(a[i].a, b[i].a);
14
15     FFT(c, n, 1);
16
17     for (int i = 0; i < n; i++) {
18         int j = (n - i) & (n - 1);
19
20         a[i] = (c[i] + c[j].conj() * 0.5;
21         b[i] = (c[i] - c[j].conj() * -0.5 * ima;
22     }
23
24 void IDFT(Complex *a, Complex *b, int n) {
25     static Complex c[maxn];
26
27     for (int i = 0; i < n; i++)
28         c[i] = a[i] + ima * b[i];
29
30
31
32
33
34
35
36
37
38
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     printf(" ");
97
98     printf("%d", ans[i]);
99
100    return 0;
101}

```

1.1.4 多项式操作

```

1 // A为输入, C为输出, n为所需长度且必须是2^k
2 // 多项式求逆, 要求A常数项不为0
3 void get_inv(int *A, int *C, int n) {
4     static int B[maxn];
5
6     memset(C, 0, sizeof(int) * (n * 2));
7     C[0] = qpow(A[0], p - 2); // 一般常数项都是1, 直接赋
→ 值为1就可以
8
9     for (int k = 2; k <= n; k *= 2) {
10        memcpy(B, A, sizeof(int) * k);
11        memset(B + k, 0, sizeof(int) * k);
12
13        NTT(B, k * 2, 1);
14        NTT(C, k * 2, 1);
15
16        for (int i = 0; i < k * 2; i++) {
17            C[i] = (2 - (long long)B[i] * C[i]) % p *
→ C[i] % p;
18            if (C[i] < 0)
19                C[i] += p;
20        }
21
22        NTT(C, k * 2, -1);
23
24        memset(C + k, 0, sizeof(int) * k);
25    }
26
27 // 开根
28 void get_sqrt(int *A, int *C, int n) {
29     static int B[maxn], D[maxn];
30
31     memset(C, 0, sizeof(int) * (n * 2));
32     C[0] = 1; // 如果不是1就要考虑二次剩余
33
34     for (int k = 2; k <= n; k *= 2) {
35        memcpy(B, A, sizeof(int) * k);
36        memset(B + k, 0, sizeof(int) * k);
37
38        get_inv(C, D, k);
39
40        NTT(B, k * 2, 1);
41        NTT(D, k * 2, 1);
42
43        for (int i = 0; i < k * 2; i++)
44            B[i] = (long long)B[i] * D[i] % p;
45
46        NTT(B, k * 2, -1);
47
48        for (int i = 0; i < k; i++)
49            C[i] = (long long)(C[i] + B[i]) * inv_2 %
→ p; // inv_2是2的逆元
50    }
51
52 // 求导
53 void get_derivative(int *A, int *C, int n) {
54     for (int i = 1; i < n; i++)

```

```

57         C[i - 1] = (long long)A[i] * i % p;
58
59     C[n - 1] = 0;
60 }
61
62 // 不定积分, 最好预处理逆元
63 void get_integrate(int *A, int *C, int n) {
64     for (int i = 1; i < n; i++)
65         C[i] = (long long)A[i - 1] * qpow(i, p - 2) %
→ p;
66
67     C[0] = 0; // 不定积分没有常数项
68 }
69
70 // 多项式ln, 要求A常数项不为0
71 void get_ln(int *A, int *C, int n) { // 通常情况下A常数
→ 项都是1
72     static int B[maxn];
73
74     get_derivative(A, B, n);
75     memset(B + n, 0, sizeof(int) * n);
76
77     get_inv(A, C, n);
78
79     NTT(B, n * 2, 1);
80     NTT(C, n * 2, 1);
81
82     for (int i = 0; i < n * 2; i++)
83         B[i] = (long long)B[i] * C[i] % p;
84
85     NTT(B, n * 2, -1);
86
87     get_integrate(B, C, n);
88
89     memset(C + n, 0, sizeof(int) * n);
90 }
91
92 // 多项式exp, 要求A没有常数项
93 // 常数很大且总代码较长, 一般来说最好替换为分治FFT
94 // 分治FFT依据: 设 $G(x) = \exp F(x)$ , 则有
95 //  $\rightarrow g_i = \sum_{k=1}^{i-1} f_{i-k} * k * g_k$ 
96 void get_exp(int *A, int *C, int n) {
97     static int B[maxn];
98
99     memset(C, 0, sizeof(int) * (n * 2));
100    C[0] = 1;
101
102    for (int k = 2; k <= n; k *= 2)
103        get_ln(C, B, k);
104
105    for (int i = 0; i < k; i++) {
106        B[i] = A[i] - B[i];
107        if (B[i] < 0)
108            B[i] += p;
109    }
110    (++B[0]) %= p;
111
112    NTT(B, k * 2, 1);
113    NTT(C, k * 2, 1);
114
115    for (int i = 0; i < k * 2; i++)
116        C[i] = (long long)C[i] * B[i] % p;
117
118    NTT(C, k * 2, -1);
119
120    memset(C + k, 0, sizeof(int) * k);
121 }
122
123 // 多项式k次幂, 在A常数项不为1时需要转化

```

```

124 // 常数较大且总代码较长，在时间要求不高时最好替换为暴力快
125 // →速幂
126 void get_pow(int *A, int *C, int n, int k) {
127     static int B[maxn];
128
129     get_ln(A, B, n);
130
131     for (int i = 0; i < n; i++)
132         B[i] = (long long)B[i] * k % p;
133
134     get_exp(B, C, n);
135 }
136
137 // 多项式除法，A / B，结果输出在C
138 // A的次数为n, B的次数为m
139 void get_div(int *A, int *B, int *C, int n, int m) {
140     static int f[maxn], g[maxn], gi[maxn];
141
142     if (n < m) {
143         memset(C, 0, sizeof(int) * m);
144         return;
145     }
146
147     int N = 1;
148     while (N < (n - m + 1))
149         N *= 2;
150
151     memset(f, 0, sizeof(int) * N * 2);
152     memset(g, 0, sizeof(int) * N * 2);
153     // memset(gi, 0, sizeof(int) * N);
154
155     for (int i = 0; i < n - m + 1; i++)
156         f[i] = A[n - i - 1];
157     for (int i = 0; i < m && i < n - m + 1; i++)
158         g[i] = B[m - i - 1];
159
160     get_inv(g, gi, N);
161
162     for (int i = n - m + 1; i < N; i++)
163         gi[i] = 0;
164
165     NTT(f, N * 2, 1);
166     NTT(gi, N * 2, 1);
167
168     for (int i = 0; i < N * 2; i++)
169         f[i] = (long long)f[i] * gi[i] % p;
170
171     NTT(f, N * 2, -1);
172
173     for (int i = 0; i < n - m + 1; i++)
174         C[i] = f[n - m - i];
175
176 // 多项式取模，余数输出到C，商输出到D
177 void get_mod(int *A, int *B, int *C, int *D, int n, int
178             m) {
179     static int b[maxn], d[maxn];
180
181     if (n < m) {
182         memcpy(C, A, sizeof(int) * n);
183
184         if (D)
185             memset(D, 0, sizeof(int) * m);
186
187         return;
188     }
189
190     get_div(A, B, d, n, m);
191
192     for (int i = 0; i < n - m + 1; i++)
193         D[i] = d[i];
194
195     int N = 1;
196     while (N < n)
197         N *= 2;
198
199     memcpy(b, B, sizeof(int) * m);
200
201     NTT(b, N, 1);
202     NTT(d, N, 1);
203
204     for (int i = 0; i < N; i++)
205         b[i] = (long long)d[i] * b[i] % p;
206
207     NTT(b, N, -1);
208
209     for (int i = 0; i < m - 1; i++)
210         C[i] = (A[i] - b[i] + p) % p;
211
212     memset(b, 0, sizeof(int) * N);
213     memset(d, 0, sizeof(int) * N);
214
215 }
216
217 // 多点求值要用的数组
218 int q[maxn], ans[maxn]; // q是要代入的各个系数, ans是求
219 // →出的值
220 int tg[25][maxn * 2], tf[25][maxn]; // 辅助数组, tg是预
221 // →处理乘积
222 // tf是项数越来越少的f, tf[0]就是原来的函数
223 void pretreat(int l, int r, int k) { // 多点求值预处理
224     static int A[maxn], B[maxn];
225
226     int *g = tg[k] + l * 2;
227
228     if (r - l + 1 ≤ 200) { // 小范围暴力, 能跑得快点
229         g[0] = 1;
230
231         for (int i = l; i ≤ r; i++) {
232             for (int j = i - l + 1; j; j--) {
233                 g[j] = (g[j - 1] - (long long)g[j] *
234                     → q[i]) % p;
235                 if (g[j] < 0)
236                     g[j] += p;
237             }
238             g[0] = (long long)g[0] * (p - q[i]) % p;
239
240         }
241
242         return;
243     }
244
245     int mid = (l + r) / 2;
246
247     pretreat(l, mid, k + 1);
248     pretreat(mid + 1, r, k + 1);
249
250     if (!k)
251         return;
252
253     int N = 1;
254     while (N ≤ r - l + 1)
255         N *= 2;
256
257     int *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid
258         → + 1) * 2;
259
260     memset(A, 0, sizeof(int) * N);

```

```

257     memset(B, 0, sizeof(int) * N);
258
259     memcpy(A, gl, sizeof(int) * (mid - l + 2));
260     memcpy(B, gr, sizeof(int) * (r - mid + 1));
261
262     NTT(A, N, 1);
263     NTT(B, N, 1);
264
265     for (int i = 0; i < N; i++)
266         A[i] = (long long)A[i] * B[i] % p;
267
268     NTT(A, N, -1);
269
270     for (int i = 0; i <= r - l + 1; i++)
271         g[i] = A[i];
272 }
273
274 void solve(int l, int r, int k) { // 多项式多点求值主过
→ 程
275     int *f = tf[k];
276
277     if (r - l + 1 <= 200) {
278         for (int i = l; i <= r; i++) {
279             int x = q[i];
280
281             for (int j = r - l; ~j; j--) {
282                 ans[i] = ((long long)ans[i] * x + f[j])
283                     → % p;
284             }
285
286             return;
287         }
288
289         int mid = (l + r) / 2;
290         int *ff = tf[k + 1], *gl = tg[k + 1] + l * 2, *gr =
→ tg[k + 1] + (mid + 1) * 2;
291
292         get_mod(f, gl, ff, nullptr, r - l + 1, mid - l +
→ 2);
293         solve(l, mid, k + 1);
294
295         memset(gl, 0, sizeof(int) * (mid - l + 2));
296         memset(ff, 0, sizeof(int) * (mid - l + 1));
297
298         get_mod(f, gr, ff, nullptr, r - l + 1, r - mid +
→ 1);
299         solve(mid + 1, r, k + 1);
300
301         memset(gr, 0, sizeof(int) * (r - mid + 1));
302         memset(ff, 0, sizeof(int) * (r - mid));
303     }
304 // f < x^n, m个询问, 询问是0-based, 当然改成1-based也很
→ 简单
305 void get_value(int *f, int *x, int *a, int n, int m) {
306     if (m <= n)
307         m = n + 1;
308     if (n < m - 1)
309         n = m - 1; // 补零方便处理
310
311     memcpy(tf[0], f, sizeof(int) * n);
312     memcpy(q, x, sizeof(int) * m);
313
314     pretreat(0, m - 1, 0);
315     solve(0, m - 1, 0);
316
317     if (a) // 如果a是nullptr, 代表不复制答案, 直接
→ 用ans数组
318     memcpy(a, ans, sizeof(int) * m);

```

319 }

1.1.5 更优秀的多项式多点求值

这个做法不需要写取模, 求逆也只有一次, 但是神乎其技, 完全搞不懂原理.

清空和复制之类的地方容易抄错, 抄的时候要注意.

```

1 int q[maxn], ans[maxn]; // q是要代入的各个系数, ans是求
→ 出的值
2 int tg[25][maxn * 2], tf[25][maxn]; // 辅助数组, tg是预
→ 处理乘积,
3 // tf是项数越来越少的f, tf[0]就是原来的函数
4
5 void pretreat(int l, int r, int k) { // 预处理
6     static int A[maxn], B[maxn];
7
8     int *g = tg[k] + l * 2;
9
10    if (r - l + 1 <= 1) { // 小范围暴力
11        g[0] = 1;
12
13        for (int i = l; i <= r; i++) {
14            for (int j = i - l + 1; j; j--) {
15                g[j] = (g[j - 1] - (long long)g[j] *
→ q[i]) % p;
16                if (g[j] < 0)
17                    g[j] += p;
18            }
19            g[0] = (long long)g[0] * (p - q[i]) % p;
20        }
21
22        reverse(g, g + r - l + 2);
23
24        return;
25    }
26
27    int mid = (l + r) / 2;
28
29    pretreat(l, mid, k + 1);
30    pretreat(mid + 1, r, k + 1);
31
32    int N = 1;
33    while (N <= r - l + 1)
34        N *= 2;
35
36    int *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid
→ + 1) * 2;
37
38    memset(A, 0, sizeof(int) * N);
39    memset(B, 0, sizeof(int) * N);
40
41    memcpy(A, gl, sizeof(int) * (mid - l + 2));
42    memcpy(B, gr, sizeof(int) * (r - mid + 1));
43
44    NTT(A, N, 1);
45    NTT(B, N, 1);
46
47    for (int i = 0; i < N; i++)
48        A[i] = (long long)A[i] * B[i] % p;
49
50    NTT(A, N, -1);
51
52    for (int i = 0; i <= r - l + 1; i++)
53        g[i] = A[i];
54
55
56 void solve(int l, int r, int k) { // 主过程
57     static int a[maxn], b[maxn];
58

```

```

59     int *f = tf[k];
60
61     if (l == r) {
62         ans[l] = f[0];
63         return;
64     }
65
66     int mid = (l + r) / 2;
67     int *ff = tf[k + 1], *gl = tg[k + 1] + l * 2, *gr =
68     ↪ tg[k + 1] + (mid + 1) * 2;
69
70     int N = 1;
71     while (N < r - l + 2)
72         N *= 2;
73
74     memcpy(a, f, sizeof(int) * (r - l + 2));
75     memcpy(b, gr, sizeof(int) * (r - mid + 1));
76     reverse(b, b + r - mid + 1);
77
78     NTT(a, N, 1);
79     NTT(b, N, 1);
80     for (int i = 0; i < N; i++)
81         b[i] = (long long)a[i] * b[i] % p;
82
83     reverse(b + 1, b + N);
84     NTT(b, N, 1);
85     int n_inv = qpow(N, p - 2);
86     for (int i = 0; i < N; i++)
87         b[i] = (long long)b[i] * n_inv % p;
88
89     for (int i = 0; i < mid - l + 2; i++)
90         ff[i] = b[i + r - mid];
91
92     memset(a, 0, sizeof(int) * N);
93     memset(b, 0, sizeof(int) * N);
94
95     solve(l, mid, k + 1);
96
97     memset(ff, 0, sizeof(int) * (mid - l + 2));
98
99     memcpy(a, f, sizeof(int) * (r - l + 2));
100    memcpy(b, gl, sizeof(int) * (mid - l + 2));
101    reverse(b, b + mid - l + 2);
102
103    NTT(a, N, 1);
104    NTT(b, N, 1);
105    for (int i = 0; i < N; i++)
106        b[i] = (long long)a[i] * b[i] % p;
107
108    reverse(b + 1, b + N);
109    NTT(b, N, 1);
110    for (int i = 0; i < N; i++)
111        b[i] = (long long)b[i] * n_inv % p;
112
113    for (int i = 0; i < r - mid + 1; i++)
114        ff[i] = b[i + mid - l + 1];
115
116    memset(a, 0, sizeof(int) * N);
117    memset(b, 0, sizeof(int) * N);
118
119    solve(mid + 1, r, k + 1);
120
121    memset(gl, 0, sizeof(int) * (mid - l + 2));
122    memset(gr, 0, sizeof(int) * (r - mid + 1));
123    memset(ff, 0, sizeof(int) * (r - mid + 1));
124
125 // f < x^n, m个询问, θ-based
126 void get_value(int *f, int *x, int *a, int n, int m) {

```

```

127     static int c[maxn], d[maxn];
128
129     if (m ≤ n)
130         m = n + 1;
131     if (n < m - 1)
132         n = m - 1; // 补零
133
134     memcpy(q, x, sizeof(int) * m);
135     pretreat(0, m - 1, 0);
136
137     int N = 1;
138     while (N < m)
139         N *= 2;
140
141     get_inv(tg[0], c, N);
142
143     fill(c + m, c + N, 0);
144     reverse(c, c + m);
145
146     memcpy(d, f, sizeof(int) * m);
147
148     NTT(c, N * 2, 1);
149     NTT(d, N * 2, 1);
150     for (int i = 0; i < N * 2; i++)
151         c[i] = (long long)c[i] * d[i] % p;
152     NTT(c, N * 2, -1);
153
154     for (int i = 0; i < m; i++)
155         tf[0][i] = c[i + n];
156
157     solve(0, m - 1, 0);
158
159     if (a) // 如果a是nullptr, 代表不复制答案, 直接
160     ↪ 用ans数组
161         memcpy(a, ans, sizeof(int) * m);
162

```

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

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

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存的是各阶数
3   → 的M(x)
4 void pretreat2(int l, int r, int k) { // 预处理
5   static int A[maxn], B[maxn];
6   int *h = th[k] + l * 2;
7
8   if (l == r) {
9     h[0] = p - qx[l];
10    h[1] = 1;
11    return;
12  }
13
14  int mid = (l + r) / 2;
15
16  pretreat2(l, mid, k + 1);
17  pretreat2(mid + 1, r, k + 1);
18
19  int N = 1;
20  while (N ≤ r - l + 1)
21    N *= 2;
22
23  int *hl = th[k + 1] + l * 2, *hr = th[k + 1] +
24    → + 1) * 2;
25
26  memset(A, 0, sizeof(int) * N);
27  memset(B, 0, sizeof(int) * N);
28
29  memcpy(A, hl, sizeof(int) * (mid - l + 2));
30  memcpy(B, hr, sizeof(int) * (r - mid + 1));
31
32  NTT(A, N, 1);
33  NTT(B, N, 1);
34
35  for (int i = 0; i < N; i++)
36    A[i] = (long long)A[i] * B[i] % p;
37
38  NTT(A, N, -1);
39
40  for (int i = 0; i ≤ r - l + 1; i++)
41    h[i] = A[i];
42
43 void solve2(int l, int r, int k) { // 分治
44   static int A[maxn], B[maxn], t[maxn];

```

```

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    pretreat2(0, n - 1, 0);
103
104    get_derivative(th[0], d, n + 1);
105
106    multipoint_eval(d, qx, nullptr, n, n);
107
108    for (int i = 0; i < n; i++)
109        ansf[i] = (long long)qy[i] * qpow(ans[i], p -
110        ↪ 2) % p;
111
112    solve2(0, n - 1, 0);
113
114    if (f)
115        memcpy(f, ansf, sizeof(int) * n);

```

113 }

1.1.7 拉格朗日反演(多项式复合逆)

如果 $f(x)$ 与 $g(x)$ 互为复合逆, 则有

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

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

1.1.8 分治FFT

```

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);
32 }
```

1.1.9 半在线卷积

```

1 void solve(int l, int r) {
2     if (r <= m)
3         return;
4
5     if (r - l == 1) {
6         if (l == m)
7             f[l] = a[m];
8         else
9             f[l] = (long long)f[l] * inv[l - m] % p;
10
11     for (int i = l, t = (long long)l * f[l] % p; i
12         → ≤ n; i += l)
13         g[i] = (g[i] + t) % p;
14
15     return;
16 }
17
18     int mid = (l + r) / 2;
19
20     solve(l, mid);
21 }
```

```

21
22     if (l == 0) {
23         for (int i = 1; i < mid; i++) {
24             A[i] = f[i];
25             B[i] = (c[i] + g[i]) % p;
26         }
27         NTT(A, r, 1);
28         NTT(B, r, 1);
29         for (int i = 0; i < r; i++)
30             A[i] = (long long)A[i] * B[i] % p;
31         NTT(A, r, -1);
32
33         for (int i = mid; i < r; i++)
34             f[i] = (f[i] + A[i]) % p;
35     }
36     else {
37         for (int i = 0; i < r - l; i++) {
38             A[i] = f[i];
39             for (int i = l; i < mid; i++)
40                 B[i - l] = (c[i] + g[i]) % p;
41         NTT(A, r - l, 1);
42         NTT(B, r - l, 1);
43         for (int i = 0; i < r - l; i++)
44             A[i] = (long long)A[i] * B[i] % p;
45         NTT(A, r - l, -1);
46
47         for (int i = mid; i < r; i++)
48             f[i] = (f[i] + A[i - l]) % p;
49
50         memset(A, 0, sizeof(int) * (r - l));
51         memset(B, 0, sizeof(int) * (r - l));
52
53         for (int i = l; i < mid; i++) {
54             A[i - l] = f[i];
55             for (int i = 0; i < r - l; i++)
56                 B[i] = (c[i] + g[i]) % p;
57         NTT(A, r - l, 1);
58         NTT(B, r - l, 1);
59         for (int i = 0; i < r - l; i++)
60             A[i] = (long long)A[i] * B[i] % p;
61         NTT(A, r - l, -1);
62
63         for (int i = mid; i < r; i++)
64             f[i] = (f[i] + A[i - l]) % p;
65     }
66
67     memset(A, 0, sizeof(int) * (r - l));
68     memset(B, 0, sizeof(int) * (r - l));
69
70     solve(mid, r);
71 }
```

1.1.10 常系数齐次线性递推 $O(k \log k \log n)$

如果只有一次这个操作可以照抄, 否则就开一个全局flag.

```

1 // 多项式取模, 余数输出到C, 商输出到D
2 void get_mod(int *A, int *B, int *C, int *D, int n, int
3 → m) {
4     static int b[maxn], d[maxn];
5     static bool flag = false;
6
7     if (n < m) {
8         memcpy(C, A, sizeof(int) * n);
9
10     if (D)
11         memset(D, 0, sizeof(int) * m);
12
13     return;
14 }
```

```

14
15     get_div(A, B, d, n, m);
16
17     if (D) { // D是商, 可以选择不要
18         for (int i = 0; i < n - m + 1; i++)
19             D[i] = d[i];
20     }
21
22     int N = 1;
23     while (N < n)
24         N *= 2;
25
26     if (!flag) {
27         memcpy(b, B, sizeof(int) * m);
28         NTT(b, N, 1);
29
30         flag = true;
31     }
32
33     NTT(d, N, 1);
34
35     for (int i = 0; i < N; i++)
36         d[i] = (long long)d[i] * b[i] % p;
37
38     NTT(d, N, -1);
39
40     for (int i = 0; i < m - 1; i++)
41         C[i] = (A[i] - d[i] + p) % p;
42
43     // memset(b, 0, sizeof(int) * N);
44     memset(d, 0, sizeof(int) * N);
45 }

// g < x^n, f是输出答案的数组
46 void pow_mod(long long k, int *g, int n, int *f) {
47     static int a[maxn], t[maxn];
48
49     memset(f, 0, sizeof(int) * (n * 2));
50
51     f[0] = a[1] = 1;
52
53     int N = 1;
54     while (N < n * 2 - 1)
55         N *= 2;
56
57     while (k) {
58         NTT(a, N, 1);
59
60         if (k & 1) {
61             memcpy(t, f, sizeof(int) * N);
62
63             NTT(t, N, 1);
64             for (int i = 0; i < N; i++)
65                 t[i] = (long long)t[i] * a[i] % p;
66             NTT(t, N, -1);
67
68             get_mod(t, g, f, NULL, n * 2 - 1, n);
69         }
70
71         for (int i = 0; i < N; i++)
72             a[i] = (long long)a[i] * a[i] % p;
73         NTT(a, N, -1);
74
75         memcpy(t, a, sizeof(int) * (n * 2 - 1));
76         get_mod(t, g, a, NULL, n * 2 - 1, n);
77         fill(a + n - 1, a + N, 0);
78
79         k >>= 1;
80     }
81
82 }

```

```

84     memset(a, 0, sizeof(int) * (n * 2));
85 }
86
87 // f_n = \sum_{i=1}^m f_{n-i} a_i
88 // f是0~m-1项的初值
89 int linear_recurrence(long long n, int m, int *f, int
90     *a) {
91     static int g[maxn], c[maxn];
92
93     memset(g, 0, sizeof(int) * (m * 2 + 1));
94
95     for (int i = 0; i < m; i++)
96         g[i] = (p - a[m - i]) % p;
97     g[m] = 1;
98
99     pow_mod(n, g, m + 1, c);
100
101    int ans = 0;
102    for (int i = 0; i < m; i++)
103        ans = (ans + (long long)c[i] * f[i]) % p;
104
105    return ans;
}

```

1.1.11 应用: $O(\sqrt{n} \log^2 n)$ 快速求阶乘

问题: 求 $n! \pmod{p}$, $n < p$, p 是 NTT 模数.

考虑令 $m = \lfloor \sqrt{n} \rfloor$, 那么我们可以写出连续 m 个数相乘的多项式:

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

那么显然就有

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

$f(x)$ 的系数可以用倍增求(或者懒一点直接分治FFT), 然后 $f(km)$ 可以用多项式多点求值求出, 所以总复杂度就是 $O(\sqrt{n} \log^2 n)$.

当然如果 p 不变并且多次询问的话我们只需要取一个 m , 也就是预处理 $O(\sqrt{p} \log^2 p)$, 询问 $O(\sqrt{p})$.

1.2 插值

1.2.1 牛顿插值

牛顿插值的原理是二项式反演.

二项式反演:

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

可以用 e^x 和 e^{-x} 的麦克劳林展开式证明.

套用二项式反演的结论即可得到牛顿插值:

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

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

其中 k 表示 $f(n)$ 的最高次项系数.

实现时可以用 k 次差分替代右边的式子:

```

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

```

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

1.2.2 拉格朗日(Lagrange)插值

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

1.3 FWT快速沃尔什变换

```

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

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

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

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

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

```

1.3.1 三行FWT

```

1 void fwt_or(int *a, int n, int tp) {
2     for (int j = 0; (1 << j) < n; j++)
3         for (int i = 0; i < n; i++)
4             if (i >> j & 1) {
5                 if (tp > 0)
6                     a[i] += a[i ^ (1 << j)];
7                 else
8                     a[i] -= a[i ^ (1 << j)];
9             }
10
11 // and自然就是or反过来
12 void fwt_and(int *a, int n, int tp) {
13     for (int j = 0; (1 << j) < n; j++)
14         for (int i = 0; i < n; i++)
15             if (!(i >> j & 1)) {
16                 if (tp > 0)
17                     a[i] += a[i | (1 << j)];
18                 else
19                     a[i] -= a[i | (1 << j)];
20             }
21
22 }
23 // xor同理

```

1.4 线性代数

稀疏矩阵操作参见Berlekamp-Massey算法的应用(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
8         if (ii != i) // 这里没有判是否无解, 如果有可能无
9             // 解的话要判一下
10            for (int j = i; j <= n + 1; j++)
11                swap(A[i][j], A[ii][j]);
12
13         for (int j = 1; j <= n; j++)
14             if (j != i) // 消成对角
15                 for (int k = n + 1; k >= i; k--)
16                     A[j][k] -= A[j][i] / A[i][i] * A[i]
17                         ^ [k];
18     }
19 }

```

解线性方程组 在矩阵的右边加上一列表示系数即可, 如果消成上三角的话最后要倒序回代.

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

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

1.4.3 行列式取模

```

1 int p;
2
3 int Gauss(int A[maxn][maxn], int n) {
4     int det = 1;
5
6     for (int i = 1; i <= n; i++) {
7         for (int j = i + 1; j <= n; j++)
8             while (A[j][i]) {
9                 int t = (p - A[i][i] / A[j][i]) % p;
10                for (int k = i; k <= n; k++)
11                    A[i][k] = (A[i][k] + (long
12                        ↪ long)A[j][k] * t) % p;
13
14                swap(A[i], A[j]);
15                det = (p - det) % p; // 交换一次之后行列
16                ↪ 式取负
17            }
18
19            if (!A[i][i])
20                return 0;
21
22            det = (long long)det * A[i][i] % p;
23        }
24
25    return det;
26 }
```

1.4.4 线性基(消成对角)

```

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

1.4.5 线性代数知识

行列式:

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

逆矩阵:

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

代数余子式:

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

也就是 A 去掉一行一列之后的行列式.

伴随矩阵:

$$A^* = C^T$$

即代数余子式矩阵的转置.

同时我们有

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

特征多项式:

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

特征根: 特征多项式的所有 n 个根(可能有重根).

1.4.6 矩阵树定理, BEST定理

无向图 设图 G 的基尔霍夫矩阵 $L(G)$ 等于度数矩阵减去邻接矩阵, 则 G 的生成树个数等于 $L(G)$ 的任意一个代数余子式的值.

有向图 类似地定义 $L_{in}(G)$ 等于入度矩阵减去邻接矩阵(i 指向 j 有边, 则 $A_{i,j} = 1$), $L_{out}(G)$ 等于出度矩阵减去邻接矩阵.

则以 i 为根的内向树个数即为 L_{out} 的第 i 个主子式(即关于第 i 行第 i 列的余子式), 外向树个数即为 L_{in} 的第 i 个主子式.

(可以看出, 只有无向图才满足 $L(G)$ 的所有代数余子式都相等.)

BEST定理(有向图欧拉回路计数) 如果 G 是有向欧拉图, 则 G 的欧拉回路的个数等于以一个任意点为根的内/外向树个数乘以 $\prod_v (\deg(v) - 1)!$.

并且在欧拉图里, 无论以哪个结点为根, 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 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.)

```

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             v = vector<int>(i + 1);
19
20             continue;
21         }
22
23         vector<int> u = v;
24         int val = (long long)(a[i] - tmp + p) *
25             ↪ qpow(delta, p - 2) % p;
26
27         if (v.size() < last.size() + i - k)
28             v.resize(last.size() + i - k);
```

```

27     v[i - k - 1] += val) %= p;
28
29     for (int j = 0; j < (int)last.size(); j++) {
30         v[i - k + j] = (v[i - k + j] - (long)
31             → long)val * last[j]) % p;
32         if (v[i - k + j] < 0)
33             v[i - k + j] += p;
34     }
35
36     if ((int)u.size() - i < (int)last.size() - k) {
37         last = u;
38         k = i;
39         delta = a[i] - tmp;
40         if (delta < 0)
41             delta += p;
42     }
43
44
45     for (auto &x : v) // 一般是需要最小递推式的, 所以处理
46         ← 一下
47         x = (p - x) % p;
48     v.insert(v.begin(), 1);
49
50     return v; // ∀i, ∑_{j=0}^m a_{i-j}v_j = 0
}

```

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.

1.5.1 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(8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.)

1.5.2 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 $O(kn + n^2)$.

1.5.3 求稀疏矩阵的行列式

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

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

设 A 有 k 个非零项, 则复杂度为 $O(kn + n^2)$.

1.5.4 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 $QAP^{-1}Q^T$ 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 $O(kn + n^2)$.

1.5.5 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    }

```

1.6 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
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    }
37
38    return 0;
39 }
```

```
38 //初始化
39 //对于初始解可行的问题,可以把初始化省略掉
40
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 || 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    }

```

1.6.1 线性规划对偶原理

给定一个原始线性规划：

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

定义它的对偶线性规划为：

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

用矩阵可以更形象地表示为：

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

1.7 博弈论

1.7.1 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1.7.3.经典博弈(14页)。

1.7.2 纳什均衡

纯策略，混合策略 纯策略是指你一定会选择某个选项，混合策略是指你对每个选项都有一个概率分布 p_i ，你会以相应的概率选择这个选项。

考虑这样的游戏：有几个人（当然也可以是两个）各自独立地做决定，然后同时公布每个人的决定，而每个人的收益和所有人的选择有关。

那么纳什均衡就是每个人都决定一个混合策略，使得在其他人都是纯策略的情况下，这个人最坏情况下（也就是说其他人的纯策略最针对他的时候）的收益是最大的。也就是说，收益函数对这个人的混合策略求一个偏导，结果是0（因为是极大值）。

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。

1.7.3 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），把任意一个石子移到低一层。

结论 奇数层的石子数量进行异或和即可。

实际上只要路径长度唯一就可以，比如在树上博弈，然后石子向根节点方向移动，那么就是奇数深度的石子数量进行异或和。

2. 可以同时操作多个子游戏

如果某个游戏由若干个独立的子游戏组成，并且每次可以任意选几个（当然至少一个）子游戏进行操作，那么结论是：所有子游戏都必败时先手才会必败，否则先手必胜。

3. 每次最多操作k个子游戏(Nim-K)

如果每次最多操作 k 个子游戏，结论是：把所有子游戏的SG函数写成二进制表示，如果每一位上的1个数都是 $(k+1)$ 的倍数，则先手必败，否则先手必胜。

（实际上上面一条可以看做 $k = \infty$ 的情况，也就是所有SG值都是0时才会先手必败。）

如果要求整个游戏的SG函数，就按照上面的方法每个二进制位相加后 $\text{mod}(k+1)$ ，视为 $(k+1)$ 进制数后求值即可。（未验证）

4. 反Nim游戏(Anti-Nim)

和Nim游戏差不多，唯一的不同是取走最后一个石子的输。

分两种情况：

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

当然石子个数实际上就是SG函数，所以判别条件全都改成SG函数也是一样的。

5. 威佐夫博弈

有两堆石子，每次要么从一堆中取任意个，要么从两堆中都取走相同数量。也等价于两个人移动一个只能向左上方走的皇后，不能动的输。

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

6. 删子树博弈

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

结论

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

7. 无向图游戏

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

结论 如果某个点一定在最大匹配中，则先手必胜，否则先手必败。

1.7.4 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，那么只需要看左上角的那个棋子。

1.8 自适应Simpson积分

Forked from fstqwq's template.

```

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

1.9 常见数列

查表参见8.15.OEIS(94页)。

1.9.1 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 }
```

1.9.2 伯努利数, 自然数幂次和

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

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

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

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

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

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

1.9.3 分拆数

```

1 int b = sqrt(n);
2 ans[0] = tmp[0] = 1;
3
4 for (int i = 1; i <= b; ++i) {
5     for (int rep = 0; rep < 2; ++rep)
6         for (int j = i; j <= n - i * i; ++j)
7             add(tmp[j], tmp[j - i]);
8
9     for (int j = i * i; j <= n; ++j)
10        add(ans[j], tmp[j - i * i]);
11 }
12
13 // —
14
15 long long a[100010];
16 long long p[50005]; // 欧拉五边形数定理
17
18 int main() {
```

```

19     p[0] = 1;
20     p[1] = 1;
21     p[2] = 2;
22     int i;
23     for (i = 1; i < 50005; i++) { // 递推式系
24         a[2 * i] = i * (i * 3 - 1) / 2; // 五边形数
25         // 为1, 5, 12, 22 ... i*(3*i-1)/2
26         a[2 * i + 1] = i * (i * 3 + 1) / 2;
27     }
28     for (i = 3; i < 50005; i++) { //
29         p[n] = p[n-1] + p[n-2] - p[n-5] -
30         p[n-7] + p[12] + p[15] - ... + p[n-i*[3i-1]/2] + p[n-
31         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) %
41             // 1000007;
42     }
43     int n;
44     while (~scanf("%d", &n))
45         printf("%lld\n", p[n]);
46 }
```

1.9.4 斯特林数

1. 第一类斯特林数

$[n]$ 表示 n 个元素划分成 k 个轮换的方案数.

递推式: $[n] = [n-1] + (n-1)[n-1]$.

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).

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

求同一列: 用一个轮换的指数生成函数做 k 次幂

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

2. 第二类斯特林数

$\{n\}$ 表示 n 个元素划分成 k 个子集的方案数.

递推式: $\{n\} = \{n-1\} + k \{n-1\}$.

求一个: 容斥, 狗都会做

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

求同一行: FFT, 狗都会做

求同一列: 指数生成函数

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

普通生成函数

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

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

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

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

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.

1.9.5 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:

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

1.9.6 欧拉数(Eulerian Number)

1. 欧拉数

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

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

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

$$\sum_{k=0}^{n-1} \begin{Bmatrix} n \\ k \end{Bmatrix} = n!$$

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

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

2. 二阶欧拉数

$\begin{Bmatrix} n \\ k \end{Bmatrix}$: 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.

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = (2n-k-1) \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + (k+1) \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

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

3. 二阶欧拉数与斯特林数的关系

$$\begin{Bmatrix} x \\ x-n \end{Bmatrix} = \sum_{k=0}^{n-1} \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+n-k-1}{2n}$$

$$\begin{Bmatrix} x \\ x-n \end{Bmatrix} = \sum_{k=0}^{n-1} \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+k}{2n}$$

1.9.7 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.

1.10 常用公式及结论

1.10.1 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: $D^2(x) = E(x^2) - E^2(x)$

1.10.2 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.

1.10.3 单位根反演(展开整除条件 $[n|k]$)

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

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

1.10.4 康托展开(排列的排名)

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

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

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

1.10.5 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 .

1.10.6 线性齐次线性常系数递推求通项

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

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

(ii). 若特征多项式

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

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

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

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



1.11 常用生成函数变换

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

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

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

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

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

2 数论

2.1 $O(n)$ 预处理逆元

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

2.2 线性筛

```
40 |  
41 |  
42 |  
43 |  
44 | { break;
```

2.3 杜教筛

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

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

```

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

```

2.4 Powerful Number筛

注意 Powerful Number 篛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

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 sqrtn;
5
6 inline int getid(long long x) {
7     if (x <= sqrtn)
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     for (int i = 1; i ≤ sqrtN; i++)
68         ans = (ans + (long long)f[i] * (g[1][getid(N /
69             → i)] - g[0][getid(N / i)])) % p;
70
71     if (ans < 0)
72         ans += p;
73
74     return ans;
75 }
76
77 int calc2(long long n) {
78     for (int j = 1; j ≤ val[0]; j++)
79         dp[j] = 1;
80
81     for (int i = 1; i ≤ prime_cnt; i++)
82         pr[i] = (pr[i - 1] + f[prime[i]]) % p;
83
84     for (int i = prime_cnt; i; i--) {
85         long long lb = (long long)prime[i] * prime[i];
86
87         for (int j = val[0]; j && val[j] ≥ lb; j--)
88             for (long long pc = prime[i]; pc ≤ val[j];
89                 → pc *= prime[i]) {
90                 int t = getid(val[j] / pc);
91
92                 int tmp = dp[t];
93                 if (r[t] > i)
94                     tmp = (tmp + pr[r[t]] - pr[max(i,
95                         → l[t])]) % p;
96
97                 dp[j] = (dp[j] + pc % p * ((pc - 1) %
98                     → p) % p * tmp) % p;
99             }
100 }
101
102 return (long long)(dp[val[0]] + pr[r[val[0]]] -
103     → pr[l[val[0]]] + p) % p;
104 }
105
106 int main() {
107
108     // ios::sync_with_stdio(false);
109
110     cin >> N;
111
112     sqrtN = (int)sqrt(N);
113
114     get_table(sqrtN);
115
116     for (int i = 1; i ≤ sqrtN; i++)
117         val[++val[0]] = i;
118
119     for (int i = 1; i ≤ sqrtN; i++)
120         val[++val[0]] = N / i;
121
122     sort(val + 1, val + val[0] + 1);
123
124     val[0] = unique(val + 1, val + val[0] + 1) - val -
125         → 1;
126
127     int li = 0, ri = 0;
128     for (int j = 1; j ≤ val[0]; j++) {
129         while (ri < prime_cnt && prime[ri + 1] ≤
130             → val[j])
131             ri++;
132
133         while (li ≤ prime_cnt && (long long)prime[li]
134             → * prime[li] ≤ val[j])
135             li++;
136     }

```

```

129         l[j] = li - 1;
130         r[j] = ri;
131     }
132
133     cout << (calc1(N) + calc2(N)) % p << endl;
134
135     return 0;
136 }

```

2.6 Miller-Rabin

```

1 // 复杂度可以认为是常数
2
3 // 封装好的函数体
4 // 需要调用check
5 bool Miller_Rabin(long long n) {
6     if (n == 1)
7         return false;
8     if (n == 2)
9         return true;
10    if (n % 2 == 0)
11        return false;
12
13    for (int i : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
14        → 31, 37}) {
15        if (i ≥ n)
16            break;
17        if (!check(n, i))
18            return false;
19    }
20
21    return true;
22}
23
24 // 用一个数检测
25 // 需要调用long long快速幂和O(1)快速乘
26 bool check(long long n, long long b) { // b: base
27     long long a = n - 1;
28     int k = 0;
29
30     while (a % 2 == 0) {
31         a /= 2;
32         k++;
33     }
34
35     long long t = qpow(b, a, n); // 这里的快速幂函数需要
36     → 写O(1)快速乘
37     if (t == 1 || t == n - 1)
38         return true;
39
40     while (k--) {
41         t = mul(t, t, n); // mul是O(1)快速乘函数
42         if (t == n - 1)
43             return true;
44     }
45
46     return false;
47 }

```

2.7 Pollard's Rho

```

1 // 注意, 虽然Pollard's Rho的理论复杂度是O( $n^{1/4}$ )的,
2 // 但实际跑起来比较慢, 一般用于做long long范围内的质因数
3 // 分解
4
5 // 封装好的函数体
6 // 需要调用solve

```

```

7 void factorize(long long n, vector<long long> &v) { // 
8     → v用于存分解出来的质因子，重复的会放多个
9     for (int i : {2, 3, 5, 7, 11, 13, 17, 19}) {
10        while (n % i == 0) {
11            v.push_back(i);
12            n /= i;
13        }
14    }
15    solve(n, v);
16    sort(v.begin(), v.end()); // 从小到大排序后返回
17}
18
19 // 递归过程
20 // 需要调用Pollard's Rho主过程，同时递归调用自身
21 void solve(long long n, vector<long long> &v) {
22     if (n == 1)
23         return;
24
25     long long p;
26     do
27         p = Pollards_Rho(n);
28     while (!p); // p是任意一个非平凡因子
29
30     if (p == n) {
31         v.push_back(p); // 说明n本身就是质数
32         return;
33     }
34
35     solve(p, v); // 递归分解两半
36     solve(n / p, v);
37
38 // Pollard's Rho主过程
39 // 需要使用Miller-Rabin作为子算法
40 // 同时需要调用O(1)快速乘和gcd函数
41 long long Pollards_Rho(long long n) {
42     // assert(n > 1);
43
44     if (Miller_Rabin(n))
45         return n;
46
47     long long c = rand() % (n - 2) + 1, i = 1, k = 2, x
48     → = rand() % (n - 3) + 2, u = 2; // 注意这里rand函
49     → 数需要重定义一下
50     while (true) {
51         i++;
52         x = (mul(x, x, n) + c) % n; // mul是O(1)快速乘函
53         → 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 快速阶乘算法

参见1.1.11.应用: $O(\sqrt{n} \log^2 n)$ 快速求阶乘(9页).

2.9 扩展欧几里德

```

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;

```

2.9.1 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.

2.9.2 类欧几里德算法(直线下整点个数)

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

```

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

```

2.10 中国剩余定理

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

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

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

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

2.10.1 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 $b = m_1 p + a_1$, $M = \text{lcm}(m_1, m_2)$.

2.11 原根阶

阶 最小的整数 k 使得 $a^k \equiv 1 \pmod{p}$, 记为 $\delta_p(a)$.

显然 a 在阶以下的幂次是两两不同的.

一个性质: 如果 a, b 均与 p 互质, 则 $\delta_p(ab) = \delta_p(a)\delta_p(b)$ 的充分必要条件是 $\gcd(\delta_p(a), \delta_p(b)) = 1$.

另外, 如果 a 与 p 互质, 则有 $\delta_p(a^k) = \frac{\delta_p(a)}{\gcd(\delta_p(a), k)}$. (也就是环上一次跳 k 步的周期.)

原根 阶等于 $\varphi(p)$ 的数.

只有形如 $2, 4, p^k, 2p^k$ (p 是奇素数) 的数才有原根, 并且如果一个数 n 有原根, 那么原根的个数是 $\varphi(\varphi(n))$ 个.

暴力找原根代码:

```

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

```

2.12 常用数论公式

2.12.1 莫比乌斯反演

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

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

2.12.2 降幂公式

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

2.12.3 其他常用公式

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

$$\mu * id = \varphi$$

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

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

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

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

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

$$\varphi * I = id$$

3 图论

3.1 最小生成树

3.1.1 Boruvka算法

思想: 每次选择连接每个连通块的最小边, 把连通块缩起来.

每次连通块个数至少减半, 所以迭代 $O(\log n)$ 次即可得到最小生成树.

一种比较简单的实现方法: 每次迭代遍历所有边, 用并查集维护连通性和每个连通块的最小边权.

应用: 最小异或生成树

3.1.2 动态最小生成树

动态最小生成树的离线算法比较容易, 而在线算法通常极为复杂.

一个跑得比较快的离线做法是对时间分治, 在每层分治时找出一定在不在MST上的边, 只带着不确定边继续递归.

简单起见, 找确定边的过程用Kruskal算法实现, 过程中的两种重要操作如下:

- Reduction: 待修改边标为 $+\infty$, 跑MST后把非树边删掉, 减少无用边
- Contraction: 待修改边标为 $-\infty$, 跑MST后缩除待修改边之外的所有MST边, 计算必须边

每轮分治需要Reduction-Contraction, 借此减少不确定边, 从而保证复杂度.

复杂度证明: 假设当前区间有 k 条待修改边, n 和 m 表示点数和边数, 那么最坏情况下R-C的效果为 $(n, m) \rightarrow (n, n + k - 1) \rightarrow (k + 1, 2k)$.

```

1 // 全局结构体与数组定义
2 struct edge { // 边的定义
3     int u, v, w, id; // id表示边在原图中的编号
4     bool vis; // 在Kruskal时用, 记录这条边是否是树边
5     bool operator < (const edge &e) const { return w <
6         → e.w; }
7 } e[20][maxn], t[maxn]; // 为了便于回滚, 在每层分治存一个
8 → 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 // 分治主过程 O(nlog^2n)
45 // 需要调用Reduction和Contraction
46 void CDQ(int l, int r, int d, int m, long long ans) {
47     ← // CDQ分治
48     if (l == r) { // 区间长度已减小到1, 输出答案, 退出
49         e[d][id[d][a[l].id]].w = a[l].v;
50         printf("%lld\n", ans + Kruskal(m, e[d]));
51         e[d][id[d][a[l].id]].w = a[l].u;
52         return;
53     }
54
55     int tmp = top;
56
57     Reduction(l, r, d, m);
58     ans += Contraction(l, r, d, m); // R-C
59
60     int mid = (l + r) / 2;
61
62     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
63     for (int i = 1; i ≤ m; i++)
64         id[d + 1][e[d][i].id] = i; // 准备好下一层要用的
65 → 数组
66
67     CDQ(l, mid, d + 1, m, ans);
68
69     for (int i = l; i ≤ mid; i++)
70         e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修
71 → 改
72
73     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
74     for (int i = 1; i ≤ m; i++)
75         id[d + 1][e[d][i].id] = i; // 重新准备下一层要用
76 → 的数组
77
78     CDQ(mid + 1, r, d + 1, m, ans);
79
80     for (int i = top; i > tmp; i--)
81         cut(stk[i]); // 撤销所有操作
82     top = tmp;
83
84 // Reduction(减少无用边): 待修改边标为 $+\infty$ , 跑MST后把非树边
85 → 删掉, 减少无用边
86 // 需要调用Kruskal
87 void Reduction(int l, int r, int d, int &m) {
88     for (int i = l; i ≤ r; i++)
89         e[d][id[d][a[i].id]].w = INF; // 待修改的边标为INF
90
91     Kruskal(m, e[d]);
92
93     copy(e[d] + 1, e[d] + m + 1, t + 1);
94
95     int cnt = 0;
96     for (int i = 1; i ≤ m; i++)
97         if (t[i].w == INF || t[i].vis) { // 非树边扔掉
98             id[d][t[i].id] = ++cnt; // 给边重新编号
99             e[d][cnt] = t[i];
99     }
99 }
```

```

100
101    for (int i = r; i >= l; i--) {
102        e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边
103        // 改回去
104    }
105
106
107
108 // Contraction(缩必须边):待修改边标为- $\text{INF}$ ,跑MST后缩除待修
109 // →改边之外的所有树边
110 // 返回缩掉的边的总权值
111 // 需要调用Kruskal
112 long long Contraction(int l, int r, int d, int &m) {
113     long long ans = 0;
114
115     for (int i = l; i <= r; i++) {
116         e[d][id[d][a[i].id]].w = - $\text{INF}$ ; // 待修改边标
117         // →为- $\text{INF}$ 
118
119         Kruskal(m, e[d]);
120         copy(e[d] + 1, e[d] + m + 1, t + 1);
121
122         int cnt = 0;
123         for (int i = 1; i <= m; i++) {
124
125             if (t[i].w != - $\text{INF}$  && t[i].vis) { // 必须边
126                 ans += t[i].w;
127                 mergeset(t[i].u, t[i].v);
128             }
129             else { // 不确定边
130                 id[d][t[i].id] = ++cnt;
131                 e[d][cnt] = t[i];
132             }
133
134         for (int i = r; i >= l; i--) {
135             e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边
136             // 改回去
137             e[d][id[d][a[i].id]].vis = false;
138
139         m = cnt;
140
141         return ans;
142
143
144 // Kruskal算法  $O(m \log n)$ 
145 // 方便起见,这里直接沿用进行过缩点的并查集,在过程结束后撤
146 // →销即可
147 long long Kruskal(int m, edge *e) {
148     int tmp = top;
149     long long ans = 0;
150
151     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过
152     // →了
153
154     for (int i = 1; i <= m; i++) {
155         if (findroot(e[i].u) != findroot(e[i].v)) {
156             e[i].vis = true;
157             ans += e[i].w;
158             mergeset(e[i].u, e[i].v);
159         }
160         else
161             e[i].vis = false;
162
163     for (int i = top; i > tmp; i--)
164         cut(stk[i]); // 撤销所有操作

```

```

164     top = tmp;
165
166     return ans;
167 }
168
169 // 以下是并查集相关函数
170 int findroot(int x) { // 因为需要撤销,不写路径压缩
171     while (p[x] != x)
172         x = p[x];
173
174     return x;
175 }
176
177 void mergeset(int x, int y) { // 按size合并,如果想跑得更
178 // →快就写一个按秩合并
179     x = findroot(x); // 但是按秩合并要再开一个栈记录合并
180     // →之前的秩
181     y = findroot(y);
182
183     if (x == y)
184         return;
185
186     if (size[x] > size[y])
187         swap(x, y);
188
189     p[x] = y;
190     size[y] += size[x];
191     stk[++top] = x;
192
193 void cut(int x) { // 并查集撤销
194     int y = x;
195
196     do
197         size[y = p[y]] -= size[x];
198     while (p[y] != y);
199
200     p[x] = x;
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
22         for (int i = 1; i <= m; i++)
23             if (mn[e[i].v] > e[i].w) {
24                 mn[e[i].v] = e[i].w;
25                 pr[e[i].v] = e[i].u;
26             }
27
28         for (int i = 1; i <= n; i++)
29             if (!alive[i]) break;
30
31         if (i == n) break;
32
33         int cur = mn[rt];
34         while (cur != pr[cur]) {
35             mn[cur] = min(mn[cur], mn[pr[cur]]);
36             cur = pr[cur];
37         }
38
39         ans += mn[rt];
40
41         for (int i = 1; i <= m; i++)
42             if (mn[e[i].v] > e[i].w) {
43                 mn[e[i].v] = e[i].w;
44                 pr[e[i].v] = e[i].u;
45             }
46
47         for (int i = 1; i <= n; i++)
48             if (!alive[i]) break;
49
50         if (i == n) break;
51
52         int cur = mn[rt];
53         while (cur != pr[cur]) {
54             mn[cur] = min(mn[cur], mn[pr[cur]]);
55             cur = pr[cur];
56         }
57
58         ans += mn[rt];
59
60         for (int i = 1; i <= m; i++)
61             if (mn[e[i].v] > e[i].w) {
62                 mn[e[i].v] = e[i].w;
63                 pr[e[i].v] = e[i].u;
64             }
65
66         for (int i = 1; i <= n; i++)
67             if (!alive[i]) break;
68
69         if (i == n) break;
70
71         int cur = mn[rt];
72         while (cur != pr[cur]) {
73             mn[cur] = min(mn[cur], mn[pr[cur]]);
74             cur = pr[cur];
75         }
76
77         ans += mn[rt];
78
79         for (int i = 1; i <= m; i++)
80             if (mn[e[i].v] > e[i].w) {
81                 mn[e[i].v] = e[i].w;
82                 pr[e[i].v] = e[i].u;
83             }
84
85         for (int i = 1; i <= n; i++)
86             if (!alive[i]) break;
87
88         if (i == n) break;
89
90         int cur = mn[rt];
91         while (cur != pr[cur]) {
92             mn[cur] = min(mn[cur], mn[pr[cur]]);
93             cur = pr[cur];
94         }
95
96         ans += mn[rt];
97
98         for (int i = 1; i <= m; i++)
99             if (mn[e[i].v] > e[i].w) {
100                mn[e[i].v] = e[i].w;
101                pr[e[i].v] = e[i].u;
102            }
103
104         for (int i = 1; i <= n; i++)
105             if (!alive[i]) break;
106
107         if (i == n) break;
108
109         int cur = mn[rt];
110         while (cur != pr[cur]) {
111             mn[cur] = min(mn[cur], mn[pr[cur]]);
112             cur = pr[cur];
113         }
114
115         ans += mn[rt];
116
117         for (int i = 1; i <= m; i++)
118             if (mn[e[i].v] > e[i].w) {
119                 mn[e[i].v] = e[i].w;
120                 pr[e[i].v] = e[i].u;
121             }
122
123         for (int i = 1; i <= n; i++)
124             if (!alive[i]) break;
125
126         if (i == n) break;
127
128         int cur = mn[rt];
129         while (cur != pr[cur]) {
130             mn[cur] = min(mn[cur], mn[pr[cur]]);
131             cur = pr[cur];
132         }
133
134         ans += mn[rt];
135
136         for (int i = 1; i <= m; i++)
137             if (mn[e[i].v] > e[i].w) {
138                 mn[e[i].v] = e[i].w;
139                 pr[e[i].v] = e[i].u;
140             }
141
142         for (int i = 1; i <= n; i++)
143             if (!alive[i]) break;
144
145         if (i == n) break;
146
147         int cur = mn[rt];
148         while (cur != pr[cur]) {
149             mn[cur] = min(mn[cur], mn[pr[cur]]);
150             cur = pr[cur];
151         }
152
153         ans += mn[rt];
154
155         for (int i = 1; i <= m; i++)
156             if (mn[e[i].v] > e[i].w) {
157                 mn[e[i].v] = e[i].w;
158                 pr[e[i].v] = e[i].u;
159             }
160
161         for (int i = 1; i <= n; i++)
162             if (!alive[i]) break;
163
164         if (i == n) break;
165
166         int cur = mn[rt];
167         while (cur != pr[cur]) {
168             mn[cur] = min(mn[cur], mn[pr[cur]]);
169             cur = pr[cur];
170         }
171
172         ans += mn[rt];
173
174         for (int i = 1; i <= m; i++)
175             if (mn[e[i].v] > e[i].w) {
176                 mn[e[i].v] = e[i].w;
177                 pr[e[i].v] = e[i].u;
178             }
179
180         for (int i = 1; i <= n; i++)
181             if (!alive[i]) break;
182
183         if (i == n) break;
184
185         int cur = mn[rt];
186         while (cur != pr[cur]) {
187             mn[cur] = min(mn[cur], mn[pr[cur]]);
188             cur = pr[cur];
189         }
190
191         ans += mn[rt];
192
193         for (int i = 1; i <= m; i++)
194             if (mn[e[i].v] > e[i].w) {
195                 mn[e[i].v] = e[i].w;
196                 pr[e[i].v] = e[i].u;
197             }
198
199         for (int i = 1; i <= n; i++)
200             if (!alive[i]) break;
201
202         if (i == n) break;
203
204         int cur = mn[rt];
205         while (cur != pr[cur]) {
206             mn[cur] = min(mn[cur], mn[pr[cur]]);
207             cur = pr[cur];
208         }
209
210         ans += mn[rt];
211
212         for (int i = 1; i <= m; i++)
213             if (mn[e[i].v] > e[i].w) {
214                 mn[e[i].v] = e[i].w;
215                 pr[e[i].v] = e[i].u;
216             }
217
218         for (int i = 1; i <= n; i++)
219             if (!alive[i]) break;
220
221         if (i == n) break;
222
223         int cur = mn[rt];
224         while (cur != pr[cur]) {
225             mn[cur] = min(mn[cur], mn[pr[cur]]);
226             cur = pr[cur];
227         }
228
229         ans += mn[rt];
230
231         for (int i = 1; i <= m; i++)
232             if (mn[e[i].v] > e[i].w) {
233                 mn[e[i].v] = e[i].w;
234                 pr[e[i].v] = e[i].u;
235             }
236
237         for (int i = 1; i <= n; i++)
238             if (!alive[i]) break;
239
240         if (i == n) break;
241
242         int cur = mn[rt];
243         while (cur != pr[cur]) {
244             mn[cur] = min(mn[cur], mn[pr[cur]]);
245             cur = pr[cur];
246         }
247
248         ans += mn[rt];
249
250         for (int i = 1; i <= m; i++)
251             if (mn[e[i].v] > e[i].w) {
252                 mn[e[i].v] = e[i].w;
253                 pr[e[i].v] = e[i].u;
254             }
255
256         for (int i = 1; i <= n; i++)
257             if (!alive[i]) break;
258
259         if (i == n) break;
260
261         int cur = mn[rt];
262         while (cur != pr[cur]) {
263             mn[cur] = min(mn[cur], mn[pr[cur]]);
264             cur = pr[cur];
265         }
266
267         ans += mn[rt];
268
269         for (int i = 1; i <= m; i++)
270             if (mn[e[i].v] > e[i].w) {
271                 mn[e[i].v] = e[i].w;
272                 pr[e[i].v] = e[i].u;
273             }
274
275         for (int i = 1; i <= n; i++)
276             if (!alive[i]) break;
277
278         if (i == n) break;
279
280         int cur = mn[rt];
281         while (cur != pr[cur]) {
282             mn[cur] = min(mn[cur], mn[pr[cur]]);
283             cur = pr[cur];
284         }
285
286         ans += mn[rt];
287
288         for (int i = 1; i <= m; i++)
289             if (mn[e[i].v] > e[i].w) {
290                 mn[e[i].v] = e[i].w;
291                 pr[e[i].v] = e[i].u;
292             }
293
294         for (int i = 1; i <= n; i++)
295             if (!alive[i]) break;
296
297         if (i == n) break;
298
299         int cur = mn[rt];
300         while (cur != pr[cur]) {
301             mn[cur] = min(mn[cur], mn[pr[cur]]);
302             cur = pr[cur];
303         }
304
305         ans += mn[rt];
306
307         for (int i = 1; i <= m; i++)
308             if (mn[e[i].v] > e[i].w) {
309                 mn[e[i].v] = e[i].w;
310                 pr[e[i].v] = e[i].u;
311             }
312
313         for (int i = 1; i <= n; i++)
314             if (!alive[i]) break;
315
316         if (i == n) break;
317
318         int cur = mn[rt];
319         while (cur != pr[cur]) {
320             mn[cur] = min(mn[cur], mn[pr[cur]]);
321             cur = pr[cur];
322         }
323
324         ans += mn[rt];
325
326         for (int i = 1; i <= m; i++)
327             if (mn[e[i].v] > e[i].w) {
328                 mn[e[i].v] = e[i].w;
329                 pr[e[i].v] = e[i].u;
330             }
331
332         for (int i = 1; i <= n; i++)
333             if (!alive[i]) break;
334
335         if (i == n) break;
336
337         int cur = mn[rt];
338         while (cur != pr[cur]) {
339             mn[cur] = min(mn[cur], mn[pr[cur]]);
340             cur = pr[cur];
341         }
342
343         ans += mn[rt];
344
345         for (int i = 1; i <= m; i++)
346             if (mn[e[i].v] > e[i].w) {
347                 mn[e[i].v] = e[i].w;
348                 pr[e[i].v] = e[i].u;
349             }
350
351         for (int i = 1; i <= n; i++)
352             if (!alive[i]) break;
353
354         if (i == n) break;
355
356         int cur = mn[rt];
357         while (cur != pr[cur]) {
358             mn[cur] = min(mn[cur], mn[pr[cur]]);
359             cur = pr[cur];
360         }
361
362         ans += mn[rt];
363
364         for (int i = 1; i <= m; i++)
365             if (mn[e[i].v] > e[i].w) {
366                 mn[e[i].v] = e[i].w;
367                 pr[e[i].v] = e[i].u;
368             }
369
370         for (int i = 1; i <= n; i++)
371             if (!alive[i]) break;
372
373         if (i == n) break;
374
375         int cur = mn[rt];
376         while (cur != pr[cur]) {
377             mn[cur] = min(mn[cur], mn[pr[cur]]);
378             cur = pr[cur];
379         }
380
381         ans += mn[rt];
382
383         for (int i = 1; i <= m; i++)
384             if (mn[e[i].v] > e[i].w) {
385                 mn[e[i].v] = e[i].w;
386                 pr[e[i].v] = e[i].u;
387             }
388
389         for (int i = 1; i <= n; i++)
390             if (!alive[i]) break;
391
392         if (i == n) break;
393
394         int cur = mn[rt];
395         while (cur != pr[cur]) {
396             mn[cur] = min(mn[cur], mn[pr[cur]]);
397             cur = pr[cur];
398         }
399
400         ans += mn[rt];
401
402         for (int i = 1; i <= m; i++)
403             if (mn[e[i].v] > e[i].w) {
404                 mn[e[i].v] = e[i].w;
405                 pr[e[i].v] = e[i].u;
406             }
407
408         for (int i = 1; i <= n; i++)
409             if (!alive[i]) break;
410
411         if (i == n) break;
412
413         int cur = mn[rt];
414         while (cur != pr[cur]) {
415             mn[cur] = min(mn[cur], mn[pr[cur]]);
416             cur = pr[cur];
417         }
418
419         ans += mn[rt];
420
421         for (int i = 1; i <= m; i++)
422             if (mn[e[i].v] > e[i].w) {
423                 mn[e[i].v] = e[i].w;
424                 pr[e[i].v] = e[i].u;
425             }
426
427         for (int i = 1; i <= n; i++)
428             if (!alive[i]) break;
429
430         if (i == n) break;
431
432         int cur = mn[rt];
433         while (cur != pr[cur]) {
434             mn[cur] = min(mn[cur], mn[pr[cur]]);
435             cur = pr[cur];
436         }
437
438         ans += mn[rt];
439
440         for (int i = 1; i <= m; i++)
441             if (mn[e[i].v] > e[i].w) {
442                 mn[e[i].v] = e[i].w;
443                 pr[e[i].v] = e[i].u;
444             }
445
446         for (int i = 1; i <= n; i++)
447             if (!alive[i]) break;
448
449         if (i == n) break;
450
451         int cur = mn[rt];
452         while (cur != pr[cur]) {
453             mn[cur] = min(mn[cur], mn[pr[cur]]);
454             cur = pr[cur];
455         }
456
457         ans += mn[rt];
458
459         for (int i = 1; i <= m; i++)
460             if (mn[e[i].v] > e[i].w) {
461                 mn[e[i].v] = e[i].w;
462                 pr[e[i].v] = e[i].u;
463             }
464
465         for (int i = 1; i <= n; i++)
466             if (!alive[i]) break;
467
468         if (i == n) break;
469
470         int cur = mn[rt];
471         while (cur != pr[cur]) {
472             mn[cur] = min(mn[cur], mn[pr[cur]]);
473             cur = pr[cur];
474         }
475
476         ans += mn[rt];
477
478         for (int i = 1; i <= m; i++)
479             if (mn[e[i].v] > e[i].w) {
480                 mn[e[i].v] = e[i].w;
481                 pr[e[i].v] = e[i].u;
482             }
483
484         for (int i = 1; i <= n; i++)
485             if (!alive[i]) break;
486
487         if (i == n) break;
488
489         int cur = mn[rt];
490         while (cur != pr[cur]) {
491             mn[cur] = min(mn[cur], mn[pr[cur]]);
492             cur = pr[cur];
493         }
494
495         ans += mn[rt];
496
497         for (int i = 1; i <= m; i++)
498             if (mn[e[i].v] > e[i].w) {
499                 mn[e[i].v] = e[i].w;
500                 pr[e[i].v] = e[i].u;
501             }
502
503         for (int i = 1; i <= n; i++)
504             if (!alive[i]) break;
505
506         if (i == n) break;
507
508         int cur = mn[rt];
509         while (cur != pr[cur]) {
510             mn[cur] = min(mn[cur], mn[pr[cur]]);
511             cur = pr[cur];
512         }
513
514         ans += mn[rt];
515
516         for (int i = 1; i <= m; i++)
517             if (mn[e[i].v] > e[i].w) {
518                 mn[e[i].v] = e[i].w;
519                 pr[e[i].v] = e[i].u;
520             }
521
522         for (int i = 1; i <= n; i++)
523             if (!alive[i]) break;
524
525         if (i == n) break;
526
527         int cur = mn[rt];
528         while (cur != pr[cur]) {
529             mn[cur] = min(mn[cur], mn[pr[cur]]);
530             cur = pr[cur];
531         }
532
533         ans += mn[rt];
534
535         for (int i = 1; i <= m; i++)
536             if (mn[e[i].v] > e[i].w) {
537                 mn[e[i].v] = e[i].w;
538                 pr[e[i].v] = e[i].u;
539             }
540
541         for (int i = 1; i <= n; i++)
542             if (!alive[i]) break;
543
544         if (i == n) break;
545
546         int cur = mn[rt];
547         while (cur != pr[cur]) {
548             mn[cur] = min(mn[cur], mn[pr[cur]]);
549             cur = pr[cur];
550         }
551
552         ans += mn[rt];
553
554         for (int i = 1; i <= m; i++)
555             if (mn[e[i].v] > e[i].w) {
556                 mn[e[i].v] = e[i].w;
557                 pr[e[i].v] = e[i].u;
558             }
559
560         for (int i = 1; i <= n; i++)
561             if (!alive[i]) break;
562
563         if (i == n) break;
564
565         int cur = mn[rt];
566         while (cur != pr[cur]) {
567             mn[cur] = min(mn[cur], mn[pr[cur]]);
568             cur = pr[cur];
569         }
570
571         ans += mn[rt];
572
573         for (int i = 1; i <= m; i++)
574             if (mn[e[i].v] > e[i].w) {
575                 mn[e[i].v] = e[i].w;
576                 pr[e[i].v] = e[i].u;
577             }
578
579         for (int i = 1; i <= n; i++)
580             if (!alive[i]) break;
581
582         if (i == n) break;
583
584         int cur = mn[rt];
585         while (cur != pr[cur]) {
586             mn[cur] = min(mn[cur], mn[pr[cur]]);
587             cur = pr[cur];
588         }
589
590         ans += mn[rt];
591
592         for (int i = 1; i <= m; i++)
593             if (mn[e[i].v] > e[i].w) {
594                 mn[e[i].v] = e[i].w;
595                 pr[e[i].v] = e[i].u;
596             }
597
598         for (int i = 1; i <= n; i++)
599             if (!alive[i]) break;
600
601         if (i == n) break;
602
603         int cur = mn[rt];
604         while (cur != pr[cur]) {
605             mn[cur] = min(mn[cur], mn[pr[cur]]);
606             cur = pr[cur];
607         }
608
609         ans += mn[rt];
610
611         for (int i = 1; i <= m; i++)
612             if (mn[e[i].v] > e[i].w) {
613                 mn[e[i].v] = e[i].w;
614                 pr[e[i].v] = e[i].u;
615             }
616
617         for (int i = 1; i <= n; i++)
618             if (!alive[i]) break;
619
620         if (i == n) break;
621
622         int cur = mn[rt];
623         while (cur != pr[cur]) {
624             mn[cur] = min(mn[cur], mn[pr[cur]]);
625             cur = pr[cur];
626         }
627
628         ans += mn[rt];
629
630         for (int i = 1; i <= m; i++)
631             if (mn[e[i].v] > e[i].w) {
632                 mn[e[i].v] = e[i].w;
633                 pr[e[i].v] = e[i].u;
634             }
635
636         for (int i = 1; i <= n; i++)
637             if (!alive[i]) break;
638
639         if (i == n) break;
640
641         int cur = mn[rt];
642         while (cur != pr[cur]) {
643             mn[cur] = min(mn[cur], mn[pr[cur]]);
644             cur = pr[cur];
645         }
646
647         ans += mn[rt];
648
649         for (int i = 1; i <= m; i++)
650             if (mn[e[i].v] > e[i].w) {
651                 mn[e[i].v] = e[i].w;
652                 pr[e[i].v] = e[i].u;
653             }
654
655         for (int i = 1; i <= n; i++)
656             if (!alive[i]) break;
657
658         if (i == n) break;
659
660         int cur = mn[rt];
661         while (cur != pr[cur]) {
662             mn[cur] = min(mn[cur], mn[pr[cur]]);
663             cur = pr[cur];
664         }
665
666         ans += mn[rt];
667
668         for (int i = 1; i <= m; i++)
669             if (mn[e[i].v] > e[i].w) {
670                 mn[e[i].v] = e[i].w;
671                 pr[e[i].v] = e[i].u;
672             }
673
674         for (int i = 1; i <= n; i++)
675             if (!alive[i]) break;
676
677         if (i == n) break;
678
679         int cur = mn[rt];
680         while (cur != pr[cur]) {
681             mn[cur] = min(mn[cur], mn[pr[cur]]);
682             cur = pr[cur];
683         }
684
685         ans += mn[rt];
686
687         for (int i = 1; i <= m; i++)
688             if (mn[e[i].v] > e[i].w) {
689                 mn[e[i].v] = e[i].w;
690                 pr[e[i].v] = e[i].u;
691             }
692
693         for (int i = 1; i <= n; i++)
694             if (!alive[i]) break;
695
696         if (i == n) break;
697
698         int cur = mn[rt];
699         while (cur != pr[cur]) {
700             mn[cur] = min(mn[cur], mn[pr[cur]]);
701             cur = pr[cur];
702         }
703
704         ans += mn[rt];
705
706         for (int i = 1; i <= m; i++)
707             if (mn[e[i].v] > e[i].w) {
708                 mn[e[i].v] = e[i].w;
709                 pr[e[i].v] = e[i].u;
710             }
711
712         for (int i = 1; i <= n; i++)
713             if (!alive[i]) break;
714
715         if (i == n) break;
716
717         int cur = mn[rt];
718         while (cur != pr[cur]) {
719             mn[cur] = min(mn[cur], mn[pr[cur]]);
720             cur = pr[cur];
721         }
722
723         ans += mn[rt];
724
725         for (int i = 1; i <= m; i++)
726             if (mn[e[i].v] > e[i].w) {
727                 mn[e[i].v] = e[i].w;
728                 pr[e[i].v] = e[i].u;
729             }
730
731         for (int i = 1; i <= n; i++)
732             if (!alive[i]) break;
733
734         if (i == n) break;
735
736         int cur = mn[rt];
737         while (cur != pr[cur]) {
738             mn[cur] = min(mn[cur], mn[pr[cur]]);
739             cur = pr[cur];
740         }
741
742         ans += mn[rt];
743
744         for (int i = 1; i <= m; i++)
745             if (mn[e[i].v] > e[i].w) {
746                 mn[e[i].v] = e[i].w;
747                 pr[e[i].v] = e[i].u;
748             }
749
750         for (int i = 1; i <= n; i++)
751             if (!alive[i]) break;
752
753         if (i == n) break;
754
755         int cur = mn[rt];
756         while (cur != pr[cur]) {
757             mn[cur] = min(mn[cur], mn[pr[cur]]);
758             cur = pr[cur];
759         }
760
761         ans += mn[rt];
762
763         for (int i = 1; i <= m; i++)
764             if (mn[e[i].v] > e[i].w) {
765                 mn[e[i].v] = e[i].w;
766                 pr[e[i].v] = e[i].u;
767             }
768
769         for (int i = 1; i <= n; i++)
770             if (!alive[i]) break;
771
772         if (i == n) break;
773
774         int cur = mn[rt];
775         while (cur != pr[cur]) {
776             mn[cur] = min(mn[cur], mn[pr[cur]]);
777             cur = pr[cur];
778         }
779
780         ans += mn[rt];
781
782         for (int i = 1; i <= m; i++)
783             if (mn[e[i].v] > e[i].w) {
784                 mn[e[i].v] = e[i].w;
785                 pr[e[i].v] = e[i].u;
786             }
787
788         for (int i = 1; i <= n; i++)
789             if (!alive[i]) break;
790
791         if (i == n) break;
792
793         int cur = mn[rt];
794         while (cur != pr[cur]) {
795             mn[cur] = min(mn[cur], mn[pr[cur]]);
796             cur = pr[cur];
797         }
798
799         ans += mn[rt];
800
801         for (int i = 1; i <= m; i++)
802             if (mn[e[i].v] > e[i].w) {
803                 mn[e[i].v] = e[i].w;
804                 pr[e[i].v] = e[i].u;
805             }
806
807         for (int i = 1; i <= n; i++)
808             if (!alive[i]) break;
809
810         if (i == n) break;
811
812         int cur = mn[rt];
813         while (cur != pr[cur]) {
814             mn[cur] = min(mn[cur], mn[pr[cur]]);
815             cur = pr[cur];
816         }
817
818         ans += mn[rt];
819
820         for (int i = 1; i <= m; i++)
821             if (mn[e[i].v] > e[i].w) {
822                 mn[e[i].v] = e[i].w;
823                 pr[e[i].v] = e[i].u;
824             }
825
826         for (int i = 1; i <= n; i++)
827             if (!alive[i]) break;
828
829         if (i == n) break;
830
831         int cur = mn[rt];
832         while (cur != pr[cur]) {
833             mn[cur] = min(mn[cur], mn[pr[cur]]);
834             cur = pr[cur];
835         }
836
837         ans += mn[rt];
838
839         for (int i = 1; i <= m; i++)
840             if (mn[e[i].v] > e[i].w) {
841                 mn[e[i].v] = e[i].w;
842                 pr[e[i].v] = e[i].u;
843             }
844
845         for (int i = 1; i <= n; i++)
846             if (!alive[i]) break;
847
848         if (i == n) break;
849
850         int cur = mn[rt];
851         while (cur != pr[cur]) {
852             mn[cur] = min(mn[cur], mn[pr[cur]]);
853             cur = pr[cur];
854         }
855
856         ans += mn[rt];
857
858         for (int i = 1; i <= m; i++)
859             if (mn[e[i].v] > e[i].w) {
860                 mn[e[i].v] = e[i].w;
861                 pr[e[i].v] = e[i].u;
862             }
863
864         for (int i = 1; i <= n; i++)
865             if (!alive[i]) break;
866
867         if (i == n) break;
868
869         int cur = mn[rt];
870         while (cur != pr[cur]) {
871             mn[cur] = min(mn[cur], mn[pr[cur]]);
872             cur = pr[cur];
873         }
874
875         ans += mn[rt];
876
877         for (int i = 1; i <= m; i++)
878             if (mn[e[i].v] > e[i].w) {
879                 mn[e[i].v] = e[i].w;
880                 pr[e[i].v] = e[i].u;
881             }
882
883         for (int
```

```

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

```

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

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

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 505;
6 constexpr long long inf = 0x3f3f3f3f3f3f3f3fll;
7
8 int g[maxn][maxn], id[maxn][maxn], pr[maxn]; // g是邻接
9     → 矩阵
10    long long f[maxn][maxn], d[maxn];
11    bool vis[maxn];
12
13 vector<pair<int, int>>
14     → minimum_diameter_spanning_tree(int n) { // 1-based
15     for (int i = 1; i <= n; i++)
16         for (int j = 1; j <= n; j++)
17             g[i][j] *= 2; // 输入的边权都要乘2
18
19     memset(f, 63, sizeof(f));
20
21     for (int i = 1; i <= n; i++)
22         f[i][i] = 0;
23
24     for (int i = 1; i <= n; i++)
25         for (int j = 1; j <= n; j++)
26             if (g[i][j])
27                 f[i][j] = g[i][j];
28
29     for (int k = 1; k <= n; k++)
30         for (int i = 1; i <= n; i++)
31             for (int j = 1; j <= n; j++)
32                 f[i][j] = min(f[i][j], f[i][k] + f[k]
33                                     → [j]);
34
35     for (int i = 1; i <= n; i++) {
36         for (int j = 1; j <= n; j++)
37             id[i][j] = j; // 距离i第j近的点
38
39         sort(id[i] + 1, id[i] + n + 1, [&i] (int x, int
40             → y) {
41             return f[i][x] < f[i][y];
42         });
43     }
44
45     int o = 0;
46     long long ansv = inf; // vertex
47
48     for (int i = 1; i <= n; i++)
49         if (f[i][id[i][n]] * 2 < ansv) {
50             ansv = f[i][id[i][n]] * 2;
51             o = i;
52         }
53
54     int u = 0, v = 0;
55     long long disu = -inf, disv = -inf, anse = inf;
56
57     for (int x = 1; x <= n; x++)
58         for (int y = 1; y <= n; y++)
59             if (g[x][y]) { // 如果g[x][y] = 0说明没有边
60                 int w = g[x][y];
61
62                 for (int i = n - 1, j = n; i; i--)
63                     if (f[y][id[x][i]] > f[y][id[x]
64                                     → [j]]) {
65                         long long tmp = f[x][id[x][i]]
66                             → + f[y][id[x][j]] + w;
67                         if (tmp < anse) {
68                             anse = tmp;
69                             u = x;
70                             v = y;
71                         }
72                     }
73
74                 printf("%lld\n", min(ansv, anse) / 2); // 直径
75
76                 memset(d, 63, sizeof(d));
77
78                 if (ansv <= anse)
79                     d[o] = 0;
80                 else {
81                     d[u] = disu;
82                     d[v] = disv;
83                 }
84
85                 for (int k = 1; k <= n; k++) { // Dijkstra
86                     int x = 0;
87                     for (int i = 1; i <= n; i++)
88                         if (!vis[i] && d[i] < d[x])
89                             x = i;
90
91                     vis[x] = true;
92                     for (int y = 1; y <= n; y++)
93                         if (g[x][y] && !vis[y]) {
94                             if (d[y] > d[x] + g[x][y]) {
95                                 d[y] = d[x] + g[x][y];
96                                 pr[y] = x;
97                             }
98                         else if (d[y] == d[x] + g[x][y] &&
99                             → d[pr[y]] < d[x])
100                             pr[y] = x;
101
102                     vector<pair<int, int>> vec;
103                     for (int i = 1; i <= n; i++)
104                         if (pr[i])
105                             vec.emplace_back(i, pr[i]);
106
107                     if (ansv > anse)
108                         vec.emplace_back(u, v);
109
110                     return vec;
111     }
112
113     int main() {
114
115         int n, m;

```

```

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

```

3.2 最短路

3.2.1 Dijkstra

参见3.2.3.k短路(27页), 注意那边是求到 t 的最短路.

3.2.2 Johnson算法(负权图多源最短路)

首先前提是图没有负环.

先任选一个起点 s , 跑一遍SPFA, 计算每个点的势 $h_u = d_{s,u}$, 然后将每条边 $u \rightarrow v$ 的权值 w 修改为 $w + h[u] - h[v]$ 即可, 由最短路的性质显然修改后边权非负.

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

3.2.3 k短路

```

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

```

```

33     node *rt; // 这个答案对应的堆顶,注意可能不等于任何一个结点的堆
34
35     B(int x, node *rt, int w) : x(x), w(w), rt(rt) {}
36
37     bool operator < (const B &a) const {
38         return w + rt → w > a.w + a.rt → w;
39     }
40 }
41
42 // 全局变量和数组定义
43 vector<int> G[maxn], W[maxn], id[maxn]; // 最开始要存反
44 //→ 向图, 然后把G清空作为儿子列表
45 bool vis[maxn], used[maxe]; // used表示边是否在最短路树
46 //→ 上
47 int u[maxe], v[maxe], w[maxe]; // 存下每条边,注意是有向
48 //→ 边
49 int d[maxn], p[maxn]; // p表示最短路树上每个点的父边
50 int n, m, k, s, t; // s, t分别表示起点和终点
51
52 // 以下是主函数中较关键的部分
53 for (int i = 0; i ≤ n; i++)
54     root[i] = null; // 一定要加上!!!
55
56 // (读入&建反向图)
57 Dijkstra();
58
59 // (清空G, W, id)
60
61 for (int i = 1; i ≤ n; i++)
62     if (p[i]) {
63         used[p[i]] = true; // 在最短路树上
64         G[v[p[i]]].push_back(i);
65     }
66
67 for (int i = 1; i ≤ m; i++) {
68     w[i] = d[u[i]] - d[v[i]]; // 现在的w[i]表示这条边能
69 //→ 使路径长度增加多少
70     if (!used[i])
71         root[u[i]] = merge(root[u[i]], newnode(w[i],
72 //→ i));
73
74 dfs(t);
75
76 priority_queue<B> heap;
77 heap.push(B(s, root[s], 0)); // 初始状态是找贡献最小的边
78 //→ 加进去
79
80 printf("%d\n", d[s]); // 第1短路需要特判
81 while (--k) { // 其余k - 1短路径用二叉堆维护
82     if (heap.empty())
83         printf("-1\n");
84     else {
85         int x = heap.top().x, w = heap.top().w;
86         node *rt = heap.top().rt;
87         heap.pop();
88
89         printf("%d\n", d[s] + w + rt → w);
90
91         if (rt → lc != null || rt → rc != null)
92             heap.push(B(x, merge(rt → lc, rt → rc),
93 //→ w)); // pop掉当前边,换成另一条贡献大一
94 //→ 点的边
95         if (root[v[rt → i]] != null)
96             heap.push(B(v[rt → i], root[v[rt → i]], w
97 //→ + rt → w)); // 保留当前边,往后面再接上
98 //→ 另一条边
99     }
100 }
```

```

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

```

3.3 Tarjan算法

3.3.1 强连通分量

```

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

```

3.3.2 割点 点双

```

1 vector<int> G[maxn], bcc[maxn];
2 int dfn[maxn], low[maxn], tim = 0, bccid[maxn], bcc_cnt
3 → = 0;
4 bool iscut[maxn];
5
6 pair<int, int> stk[maxn];
7 int stk_cnt = 0;
8
9 void dfs(int x, int pr) {
10    int child = 0;
11    dfn[x] = low[x] = ++tim;
12
13    for (int y : G[x]) {
14        if (!dfn[y]) {
15            stk[++stk_cnt] = make_pair(x, y);
16            child++;
17            dfs(y, x);
18            low[x] = min(low[x], low[y]);
19
20            if (low[y] ≥ dfn[x]) {
21                iscut[x] = true;
22                bcc_cnt++;
23            }
24        }
25    }
26
27    if (child == 1)
28        iscut[x] = true;
29
30    while (true) {
31        if (stk[0] == x)
32            break;
33        pair<int, int> p = stk[0];
34        stk[0] = stk[stk_cnt];
35        stk[stk_cnt] = p;
36        stk_cnt--;
37    }
38 }
39
40

```

```

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

```

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 仙人掌

一般来说仙人掌问题都可以通过圆方树转成有两种点的树上问题来做。

3.4.1 仙人掌DP

```

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                 tmp += ww;
58
59                 if (e[i ^ 1].to == y)
60                     break;
61             }
62         }
63     }
64 }

```

```

61     }
62
63     addedge(y, cnt, 0);
64
65     sum[cnt] = tmp;
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    }

```

```

15     return true;
16 }
17
18     return false;
19 }
20
21 int hungary() {
22     int ans = 0;
23
24     for (int i = 1; i <= n; i++)
25         if (!girl[i]) {
26             memset(vis, 0, sizeof(vis));
27             ans += dfs(i);
28         }
29
30     return ans;
31 }
32

```

3.5.2 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     bool flag = false;
22
23     while (head != tail) {
24         int x = q[head++];
25
26         for (auto y : G[x])
27             if (dy[y] == -1) {
28                 dy[y] = dx[x] + 1;
29
30                 if (boy[y]) {
31                     if (dx[boy[y]] == -1) {
32                         dx[boy[y]] = dy[y] + 1;
33                         q[tail++] = boy[y];
34                     }
35                 } else
36                     flag = true;
37             }
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) {

```

3.5 二分图

3.5.1 匈牙利

```

1 vector<int> G[maxn];
2
3 int girl[maxn], boy[maxn]; // 男孩在左边，女孩在右边
4 bool vis[maxn];
5
6 bool dfs(int x) {
7     for (int y : G[x])
8         if (!vis[y]) {
9             vis[y] = true;
10
11             if (!boy[y] || dfs(boy[y])) {
12                 girl[x] = y;
13                 boy[y] = x;
14             }
15         }
16     }
17
18 }
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

```

```

46     vis[y] = true;
47
48     if (boy[y] && !dfs(boy[y]))
49         continue;
50
51     girl[x] = y;
52     boy[y] = x;
53     return true;
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
   ↪ 配, 当然不写贪心也行
62         for (int y : G[x])
63             if (!boy[y]) {
64                 girl[x] = y;
65                 boy[y] = x;
66                 ans++;
67                 break;
68             }
69
70     while (bfs(n)) {
71         memset(vis, 0, sizeof(bool) * (n + 1));
72
73         for (int x = 1; x ≤ n; x++)
74             if (!girl[x])
75                 ans += dfs(x);
76     }
77
78     return ans;
79 }
80

```

```

28     return true;
29 }

30 // bfs每个点
31 void bfs(int x) {
32     memset(q, 0, sizeof(q));
33     head = tail = 0;
34
35     q[tail++] = x;
36     visx[x] = true;
37
38     while (true) {
39         while (head != tail) {
40             int x = q[head++];
41
42             for (int y = 1; y ≤ N; y++)
43                 if (!visy[y]) {
44                     long long d = lx[x] + ly[y] - w[x]
   ↪ [y];
45
46                     if (d < slack[y]) {
47                         p[y] = x;
48                         slack[y] = d;
49
50                     if (!slack[y] && check(y))
51                         return;
52                 }
53             }
54         }
55     }
56
57     long long d = INF;
58     for (int i = 1; i ≤ N; i++)
59         if (!visy[i])
60             d = min(d, slack[i]);
61
62     for (int i = 1; i ≤ N; i++) {
63         if (visx[i])
64             lx[i] -= d;
65
66         if (visy[i])
67             ly[i] += d;
68         else
69             slack[i] -= d;
70     }
71
72     for (int i = 1; i ≤ N; i++)
73         if (!visy[i] && !slack[i] && check(i))
74             return;
75 }
76
77 // 主过程
78 long long KM() {
79     for (int i = 1; i ≤ N; i++) {
80         // lx[i] = 0;
81         ly[i] = -INF;
82         // boy[i] = girl[i] = -1;
83
84         for (int j = 1; j ≤ N; j++)
85             ly[i] = max(ly[i], w[j][i]);
86     }
87
88     for (int i = 1; i ≤ N; i++) {
89         memset(slack, 0x3f, sizeof(slack));
90         memset(visx, 0, sizeof(visx));
91         memset(visy, 0, sizeof(visy));
92         bfs(i);
93     }
94 }

```

3.5.3 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],
   ↪ slack[maxn];
4 // 边权 顶标 slack
5 // 如果要求最大权完美匹配就把不存在的边设为-INF, 否则所有
   ↪ 边对0取max
6
7 bool visx[maxn], visy[maxn];
8
9 int boy[maxn], girl[maxn], p[maxn], q[maxn], head,
   ↪ 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

```

```

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 }
```

3.5.4 二分图原理

- 最大匹配的可行边与必须边，关键点

以下的“残量网络”指网络流图的残量网络。

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

- 独立集

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

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

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

(由于一个点最多流出一个流量，所以中间的边权其实是可以任取的。)

- 二分图最大权匹配

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

显然KM算法的原理实际上就是求最小顶标和。

3.6 一般图匹配

3.6.1 高斯消元

```

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
```

```

4 // 需要输出方案时才需要再写后面那些乱七八糟的东西
5
6
7 // 复杂度和常数所限，1s之内500已经是这个算法的极限了
8 const int maxn = 505, p = 1000000007; // p可以是任意10^9以内的质数
9
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[++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
60                     eliminate(i, j);
61                     eliminate(j, i);
62                     break;
63                 }
64
65     }
66
67     printf("%d\n", n / 2);
68     for (int i = 1; i <= m; i++)
69         printf("%d ", girl[i]);
70
71     return 0;
72 }
73
74 // 高斯消元 O(n^3)
```

134		
135	{	}

3.6.2 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
5 // 全局数组与变量定义
6 vector<int> G[maxn];
7 int girl[maxn], f[maxn], t[maxn], p[maxn], vis[maxn],
8     ↳tim, q[maxn], head, tail;
9 int n, m;
10
11
12 // 封装好的主过程 O(nm)
13 int blossom() {
14     int ans = 0;
15
16     for (int i = 1; i ≤ n; i++)
17         if (!girl[i])
18             ans += bfs(i);
19
20     return ans;
21 }
22
23 // bfs找增广路 O(m)
24 bool bfs(int s) {
25     memset(t, 0, sizeof(t));
26     memset(p, 0, sizeof(p));
27
28     for (int i = 1; i ≤ n; i++)
29         f[i] = i; // 并查集
30
31     head = tail = 0;
32     q[tail++] = s;
33     t[s] = 1;
34
35     while (head != tail) {
36         int x = q[head++];
37         for (int y : G[x]) {
38             if (findroot(y) == findroot(x) || t[y] ==
39                 ↳2)
40                 continue;
41
42             if (!t[y]) {
43                 t[y] = 2;
44                 p[y] = x;
45
46                 if (!girl[y]) {
47                     for (int u = y; t[u] < t[y]; u = t[u]) {
48                         t = girl[p[u]];
49                         girl[p[u]] = u;
50                         girl[u] = p[u];
51                     }
52                     return true;
53                 }
54
55                 t[girl[y]] = 1;
56                 q[tail++] = girl[y];
57             } else if (t[y] == 1) {
58                 int z = LCA(x, y);
59
60                 shrink(x, y, z);
61                 shrink(y, x, z);
62             }
63         }
64     }
65 }

```

```

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) {
74             t[y] = 1;
75             q[tail++] = y;
76         }
77
78         if (findroot(x) == x)
79             f[x] = z;
80         if (findroot(y) == y)
81             f[y] = z;
82
83         x = p[y];
84     }
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

```

3.6.3 带权带花树

Forked from the template of Imperisble Night.

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

```

1 //maximum weight blossom, change g[u][v].w to INF -
2 //→ g[u][v].w when minimum weight blossom is needed
3 //type of ans is long long
4 //replace all int to long long if weight of edge is
5 //→ long long
6
7 struct WeightGraph {
8     static const int INF = INT_MAX;
9     static const int MAXN = 400;
10    struct edge{
11        int u, v, w;
12        edge() {}
13        edge(int u, int v, int w): u(u), v(v), w(w) {}
14    };
15
16    vector<vector<edge>> adj;
17    vector<int> p, t, f, st, S, lab, slack, flower;
18    queue<int> q;
19
20    inline int e_delta(const edge &e){ // does not work
21        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
22    }
23
24    inline void update_slack(int u, int x){
25        if(!slack[x] || e_delta(g[u][x]) <
26            e_delta(g[slack[x]][x]))
27            slack[x] = u;
28    }
29
30    inline void set_slack(int x){
31        slack[x] = 0;
32        for(int u = 1; u ≤ n; ++u)
33            if(g[u][x].w > 0 && st[u] != x && S[st[u]] ==
34                0)
35                update_slack(u, x);
36
37    void q_push(int x){
38        if(x ≤ n)q.push(x);
39        else for(size_t i = 0; i < flower[x].size(); i+
40            +)
41            q.push(flower[x][i]);
42
43    inline void set_st(int x, int b){
44        st[x]=b;
45        if(x > n) for(size_t i = 0; i <
46            flower[x].size(); ++i)
47                set_st(flower[x][i], b);
48
49    inline int get_pr(int b, int xr){
50        int pr = find(flower[b].begin(),
51            flower[b].end(), xr) - flower[b].begin();
52        if(pr % 2 == 1){
53            reverse(flower[b].begin() + 1,
54                flower[b].end());
55            return (int)flower[b].size() - pr;
56        } else return pr;
57    }
58
59    inline void set_match(int u, int v){
60        match[u]=g[u][v].v;
61        if(u > n){
62            edge e=g[u][v];
63            int xr = flower_from[u][e.u], pr=get_pr(u,
64                → xr);
65            for(int i = 0; i < pr; ++i)
66                set_match(flower[u][i], flower[u][i ^
67                    + 1]);
68            set_match(xr, v);
69            rotate(flower[u].begin(),
70                → flower[u].begin() + pr, flower[u].end());
71        }
72
73    inline void augment(int u, int v){
74        for(; ; ){
75            int xnv=st[match[u]];
76            set_match(u, v);
77            if(!xnv) return;
78            set_match(xnv, st[pa[xnv]]);
79            u=st[pa[xnv]], v=xnv;
80        }
81
82    }
83
84    inline void rotate(int u, int v, int w, int x, int y, int z){
85        if(u == v) return;
86        if(u == x) swap(u, v);
87        if(v == y) swap(v, z);
88        if(x == y) swap(x, z);
89
90        edge e1 = adj[u][v];
91        edge e2 = adj[v][z];
92        edge e3 = adj[z][y];
93        edge e4 = adj[y][x];
94
95        adj[u][v] = {v, z, w};
96        adj[v][z] = {z, y, w};
97        adj[z][y] = {y, x, w};
98        adj[y][x] = {x, u, w};
99
100    }
101
102    inline void flip(int u, int v, int w, int x, int y, int z){
103        if(u == v) return;
104        if(u == x) swap(u, v);
105        if(v == y) swap(v, z);
106        if(x == y) swap(x, z);
107
108        edge e1 = adj[u][v];
109        edge e2 = adj[v][z];
110        edge e3 = adj[z][y];
111        edge e4 = adj[y][x];
112
113        adj[u][v] = {v, z, -w};
114        adj[v][z] = {z, y, -w};
115        adj[z][y] = {y, x, -w};
116        adj[y][x] = {x, u, -w};
117
118    }
119
120    inline void flip(int u, int v, int w, int x, int y, int z){
121        if(u == v) return;
122        if(u == x) swap(u, v);
123        if(v == y) swap(v, z);
124        if(x == y) swap(x, z);
125
126        edge e1 = adj[u][v];
127        edge e2 = adj[v][z];
128        edge e3 = adj[z][y];
129        edge e4 = adj[y][x];
130
131        adj[u][v] = {v, z, -w};
132        adj[v][z] = {z, y, -w};
133        adj[z][y] = {y, x, -w};
134        adj[y][x] = {x, u, -w};
135
136    }
137
138    inline void flip(int u, int v, int w, int x, int y, int z){
139        if(u == v) return;
140        if(u == x) swap(u, v);
141        if(v == y) swap(v, z);
142        if(x == y) swap(x, z);
143
144        edge e1 = adj[u][v];
145        edge e2 = adj[v][z];
146        edge e3 = adj[z][y];
147        edge e4 = adj[y][x];
148
149        adj[u][v] = {v, z, -w};
150        adj[v][z] = {z, y, -w};
151        adj[z][y] = {y, x, -w};
152        adj[y][x] = {x, u, -w};
153
154    }
155
156    inline void flip(int u, int v, int w, int x, int y, int z){
157        if(u == v) return;
158        if(u == x) swap(u, v);
159        if(v == y) swap(v, z);
160        if(x == y) swap(x, z);
161
162        edge e1 = adj[u][v];
163        edge e2 = adj[v][z];
164        edge e3 = adj[z][y];
165        edge e4 = adj[y][x];
166
167        adj[u][v] = {v, z, -w};
168        adj[v][z] = {z, y, -w};
169        adj[z][y] = {y, x, -w};
170        adj[y][x] = {x, u, -w};
171
172    }
173
174    inline void flip(int u, int v, int w, int x, int y, int z){
175        if(u == v) return;
176        if(u == x) swap(u, v);
177        if(v == y) swap(v, z);
178        if(x == y) swap(x, z);
179
180        edge e1 = adj[u][v];
181        edge e2 = adj[v][z];
182        edge e3 = adj[z][y];
183        edge e4 = adj[y][x];
184
185        adj[u][v] = {v, z, -w};
186        adj[v][z] = {z, y, -w};
187        adj[z][y] = {y, x, -w};
188        adj[y][x] = {x, u, -w};
189
190    }
191
192    inline void flip(int u, int v, int w, int x, int y, int z){
193        if(u == v) return;
194        if(u == x) swap(u, v);
195        if(v == y) swap(v, z);
196        if(x == y) swap(x, z);
197
198        edge e1 = adj[u][v];
199        edge e2 = adj[v][z];
200        edge e3 = adj[z][y];
201        edge e4 = adj[y][x];
202
203        adj[u][v] = {v, z, -w};
204        adj[v][z] = {z, y, -w};
205        adj[z][y] = {y, x, -w};
206        adj[y][x] = {x, u, -w};
207
208    }
209
210    inline void flip(int u, int v, int w, int x, int y, int z){
211        if(u == v) return;
212        if(u == x) swap(u, v);
213        if(v == y) swap(v, z);
214        if(x == y) swap(x, z);
215
216        edge e1 = adj[u][v];
217        edge e2 = adj[v][z];
218        edge e3 = adj[z][y];
219        edge e4 = adj[y][x];
220
221        adj[u][v] = {v, z, -w};
222        adj[v][z] = {z, y, -w};
223        adj[z][y] = {y, x, -w};
224        adj[y][x] = {x, u, -w};
225
226    }
227
228    inline void flip(int u, int v, int w, int x, int y, int z){
229        if(u == v) return;
230        if(u == x) swap(u, v);
231        if(v == y) swap(v, z);
232        if(x == y) swap(x, z);
233
234        edge e1 = adj[u][v];
235        edge e2 = adj[v][z];
236        edge e3 = adj[z][y];
237        edge e4 = adj[y][x];
238
239        adj[u][v] = {v, z, -w};
240        adj[v][z] = {z, y, -w};
241        adj[z][y] = {y, x, -w};
242        adj[y][x] = {x, u, -w};
243
244    }
245
246    inline void flip(int u, int v, int w, int x, int y, int z){
247        if(u == v) return;
248        if(u == x) swap(u, v);
249        if(v == y) swap(v, z);
250        if(x == y) swap(x, z);
251
252        edge e1 = adj[u][v];
253        edge e2 = adj[v][z];
254        edge e3 = adj[z][y];
255        edge e4 = adj[y][x];
256
257        adj[u][v] = {v, z, -w};
258        adj[v][z] = {z, y, -w};
259        adj[z][y] = {y, x, -w};
260        adj[y][x] = {x, u, -w};
261
262    }
263
264    inline void flip(int u, int v, int w, int x, int y, int z){
265        if(u == v) return;
266        if(u == x) swap(u, v);
267        if(v == y) swap(v, z);
268        if(x == y) swap(x, z);
269
270        edge e1 = adj[u][v];
271        edge e2 = adj[v][z];
272        edge e3 = adj[z][y];
273        edge e4 = adj[y][x];
274
275        adj[u][v] = {v, z, -w};
276        adj[v][z] = {z, y, -w};
277        adj[z][y] = {y, x, -w};
278        adj[y][x] = {x, u, -w};
279
280    }
281
282    inline void flip(int u, int v, int w, int x, int y, int z){
283        if(u == v) return;
284        if(u == x) swap(u, v);
285        if(v == y) swap(v, z);
286        if(x == y) swap(x, z);
287
288        edge e1 = adj[u][v];
289        edge e2 = adj[v][z];
290        edge e3 = adj[z][y];
291        edge e4 = adj[y][x];
292
293        adj[u][v] = {v, z, -w};
294        adj[v][z] = {z, y, -w};
295        adj[z][y] = {y, x, -w};
296        adj[y][x] = {x, u, -w};
297
298    }
299
300    inline void flip(int u, int v, int w, int x, int y, int z){
301        if(u == v) return;
302        if(u == x) swap(u, v);
303        if(v == y) swap(v, z);
304        if(x == y) swap(x, z);
305
306        edge e1 = adj[u][v];
307        edge e2 = adj[v][z];
308        edge e3 = adj[z][y];
309        edge e4 = adj[y][x];
310
311        adj[u][v] = {v, z, -w};
312        adj[v][z] = {z, y, -w};
313        adj[z][y] = {y, x, -w};
314        adj[y][x] = {x, u, -w};
315
316    }
317
318    inline void flip(int u, int v, int w, int x, int y, int z){
319        if(u == v) return;
320        if(u == x) swap(u, v);
321        if(v == y) swap(v, z);
322        if(x == y) swap(x, z);
323
324        edge e1 = adj[u][v];
325        edge e2 = adj[v][z];
326        edge e3 = adj[z][y];
327        edge e4 = adj[y][x];
328
329        adj[u][v] = {v, z, -w};
330        adj[v][z] = {z, y, -w};
331        adj[z][y] = {y, x, -w};
332        adj[y][x] = {x, u, -w};
333
334    }
335
336    inline void flip(int u, int v, int w, int x, int y, int z){
337        if(u == v) return;
338        if(u == x) swap(u, v);
339        if(v == y) swap(v, z);
340        if(x == y) swap(x, z);
341
342        edge e1 = adj[u][v];
343        edge e2 = adj[v][z];
344        edge e3 = adj[z][y];
345        edge e4 = adj[y][x];
346
347        adj[u][v] = {v, z, -w};
348        adj[v][z] = {z, y, -w};
349        adj[z][y] = {y, x, -w};
350        adj[y][x] = {x, u, -w};
351
352    }
353
354    inline void flip(int u, int v, int w, int x, int y, int z){
355        if(u == v) return;
356        if(u == x) swap(u, v);
357        if(v == y) swap(v, z);
358        if(x == y) swap(x, z);
359
360        edge e1 = adj[u][v];
361        edge e2 = adj[v][z];
362        edge e3 = adj[z][y];
363        edge e4 = adj[y][x];
364
365        adj[u][v] = {v, z, -w};
366        adj[v][z] = {z, y, -w};
367        adj[z][y] = {y, x, -w};
368        adj[y][x] = {x, u, -w};
369
370    }
371
372    inline void flip(int u, int v, int w, int x, int y, int z){
373        if(u == v) return;
374        if(u == x) swap(u, v);
375        if(v == y) swap(v, z);
376        if(x == y) swap(x, z);
377
378        edge e1 = adj[u][v];
379        edge e2 = adj[v][z];
380        edge e3 = adj[z][y];
381        edge e4 = adj[y][x];
382
383        adj[u][v] = {v, z, -w};
384        adj[v][z] = {z, y, -w};
385        adj[z][y] = {y, x, -w};
386        adj[y][x] = {x, u, -w};
387
388    }
389
390    inline void flip(int u, int v, int w, int x, int y, int z){
391        if(u == v) return;
392        if(u == x) swap(u, v);
393        if(v == y) swap(v, z);
394        if(x == y) swap(x, z);
395
396        edge e1 = adj[u][v];
397        edge e2 = adj[v][z];
398        edge e3 = adj[z][y];
399        edge e4 = adj[y][x];
400
401        adj[u][v] = {v, z, -w};
402        adj[v][z] = {z, y, -w};
403        adj[z][y] = {y, x, -w};
404        adj[y][x] = {x, u, -w};
405
406    }
407
408    inline void flip(int u, int v, int w, int x, int y, int z){
409        if(u == v) return;
410        if(u == x) swap(u, v);
411        if(v == y) swap(v, z);
412        if(x == y) swap(x, z);
413
414        edge e1 = adj[u][v];
415        edge e2 = adj[v][z];
416        edge e3 = adj[z][y];
417        edge e4 = adj[y][x];
418
419        adj[u][v] = {v, z, -w};
420        adj[v][z] = {z, y, -w};
421        adj[z][y] = {y, x, -w};
422        adj[y][x] = {x, u, -w};
423
424    }
425
426    inline void flip(int u, int v, int w, int x, int y, int z){
427        if(u == v) return;
428        if(u == x) swap(u, v);
429        if(v == y) swap(v, z);
430        if(x == y) swap(x, z);
431
432        edge e1 = adj[u][v];
433        edge e2 = adj[v][z];
434        edge e3 = adj[z][y];
435        edge e4 = adj[y][x];
436
437        adj[u][v] = {v, z, -w};
438        adj[v][z] = {z, y, -w};
439        adj[z][y] = {y, x, -w};
440        adj[y][x] = {x, u, -w};
441
442    }
443
444    inline void flip(int u, int v, int w, int x, int y, int z){
445        if(u == v) return;
446        if(u == x) swap(u, v);
447        if(v == y) swap(v, z);
448        if(x == y) swap(x, z);
449
450        edge e1 = adj[u][v];
451        edge e2 = adj[v][z];
452        edge e3 = adj[z][y];
453        edge e4 = adj[y][x];
454
455        adj[u][v] = {v, z, -w};
456        adj[v][z] = {z, y, -w};
457        adj[z][y] = {y, x, -w};
458        adj[y][x] = {x, u, -w};
459
460    }
461
462    inline void flip(int u, int v, int w, int x, int y, int z){
463        if(u == v) return;
464        if(u == x) swap(u, v);
465        if(v == y) swap(v, z);
466        if(x == y) swap(x, z);
467
468        edge e1 = adj[u][v];
469        edge e2 = adj[v][z];
470        edge e3 = adj[z][y];
471        edge e4 = adj[y][x];
472
473        adj[u][v] = {v, z, -w};
474        adj[v][z] = {z, y, -w};
475        adj[z][y] = {y, x, -w};
476        adj[y][x] = {x, u, -w};
477
478    }
479
480    inline void flip(int u, int v, int w, int x, int y, int z){
481        if(u == v) return;
482        if(u == x) swap(u, v);
483        if(v == y) swap(v, z);
484        if(x == y) swap(x, z);
485
486        edge e1 = adj[u][v];
487        edge e2 = adj[v][z];
488        edge e3 = adj[z][y];
489        edge e4 = adj[y][x];
490
491        adj[u][v] = {v, z, -w};
492        adj[v][z] = {z, y, -w};
493        adj[z][y] = {y, x, -w};
494        adj[y][x] = {x, u, -w};
495
496    }
497
498    inline void flip(int u, int v, int w, int x, int y, int z){
499        if(u == v) return;
500        if(u == x) swap(u, v);
501        if(v == y) swap(v, z);
502        if(x == y) swap(x, z);
503
504        edge e1 = adj[u][v];
505        edge e2 = adj[v][z];
506        edge e3 = adj[z][y];
507        edge e4 = adj[y][x];
508
509        adj[u][v] = {v, z, -w};
510        adj[v][z] = {z, y, -w};
511        adj[z][y] = {y, x, -w};
512        adj[y][x] = {x, u, -w};
513
514    }
515
516    inline void flip(int u, int v, int w, int x, int y, int z){
517        if(u == v) return;
518        if(u == x) swap(u, v);
519        if(v == y) swap(v, z);
520        if(x == y) swap(x, z);
521
522        edge e1 = adj[u][v];
523        edge e2 = adj[v][z];
524        edge e3 = adj[z][y];
525        edge e4 = adj[y][x];
526
527        adj[u][v] = {v, z, -w};
528        adj[v][z] = {z, y, -w};
529        adj[z][y] = {y, x, -w};
530        adj[y][x] = {x, u, -w};
531
532    }
533
534    inline void flip(int u, int v, int w, int x, int y, int z){
535        if(u == v) return;
536        if(u == x) swap(u, v);
537        if(v == y) swap(v, z);
538        if(x == y) swap(x, z);
539
540        edge e1 = adj[u][v];
541        edge e2 = adj[v][z];
542        edge e3 = adj[z][y];
543        edge e4 = adj[y][x];
544
545        adj[u][v] = {v, z, -w};
546        adj[v][z] = {z, y, -w};
547        adj[z][y] = {y, x, -w};
548        adj[y][x] = {x, u, -w};
549
550    }
551
552    inline void flip(int u, int v, int w, int x, int y, int z){
553        if(u == v) return;
554        if(u == x) swap(u, v);
555        if(v == y) swap(v, z);
556        if(x == y) swap(x, z);
557
558        edge e1 = adj[u][v];
559        edge e2 = adj[v][z];
560        edge e3 = adj[z][y];
561        edge e4 = adj[y][x];
562
563        adj[u][v] = {v, z, -w};
564        adj[v][z] = {z, y, -w};
565        adj[z][y] = {y, x, -w};
566        adj[y][x] = {x, u, -w};
567
568    }
569
570    inline void flip(int u, int v, int w, int x, int y, int z){
571        if(u == v) return;
572        if(u == x) swap(u, v);
573        if(v == y) swap(v, z);
574        if(x == y) swap(x, z);
575
576        edge e1 = adj[u][v];
577        edge e2 = adj[v][z];
578        edge e3 = adj[z][y];
579        edge e4 = adj[y][x];
580
581        adj[u][v] = {v, z, -w};
582        adj[v][z] = {z, y, -w};
583        adj[z][y] = {y, x, -w};
584        adj[y][x] = {x, u, -w};
585
586    }
587
588    inline void flip(int u, int v, int w, int x, int y, int z){
589        if(u == v) return;
590        if(u == x) swap(u, v);
591        if(v == y) swap(v, z);
592        if(x == y) swap(x, z);
593
594        edge e1 = adj[u][v];
595        edge e2 = adj[v][z];
596        edge e3 = adj[z][y];
597        edge e4 = adj[y][x];
598
599        adj[u][v] = {v, z, -w};
600        adj[v][z] = {z, y, -w};
601        adj[z][y] = {y, x, -w};
602        adj[y][x] = {x, u, -w};
603
604    }
605
606    inline void flip(int u, int v, int w, int x, int y, int z){
607        if(u == v) return;
608        if(u == x) swap(u, v);
609        if(v == y) swap(v, z);
610        if(x == y) swap(x, z);
611
612        edge e1 = adj[u][v];
613        edge e2 = adj[v][z];
614        edge e3 = adj[z][y];
615        edge e4 = adj[y][x];
616
617        adj[u][v] = {v, z, -w};
618        adj[v][z] = {z, y, -w};
619        adj[z][y] = {y, x, -w};
620        adj[y][x] = {x, u, -w};
621
622    }
623
624    inline void flip(int u, int v, int w, int x, int y, int z){
625        if(u == v) return;
626        if(u == x) swap(u, v);
627        if(v == y) swap(v, z);
628        if(x == y) swap(x, z);
629
630        edge e1 = adj[u][v];
631        edge e2 = adj[v][z];
632        edge e3 = adj[z][y];
633        edge e4 = adj[y][x];
634
635        adj[u][v] = {v, z, -w};
636        adj[v][z] = {z, y, -w};
637        adj[z][y] = {y, x, -w};
638        adj[y][x] = {x, u, -w};
639
640    }
641
642    inline void flip(int u, int v, int w, int x, int y, int z){
643        if(u == v) return;
644        if(u == x) swap(u, v);
645        if(v == y) swap(v, z);
646        if(x == y) swap(x, z);
647
648        edge e1 = adj[u][v];
649        edge e2 = adj[v][z];
650        edge e3 = adj[z][y];
651        edge e4 = adj[y][x];
652
653        adj[u][v] = {v, z, -w};
654        adj[v][z] = {z, y, -w};
655        adj[z][y] = {y, x, -w};
656        adj[y][x] = {x, u, -w};
657
658    }
659
660    inline void flip(int u, int v, int w, int x, int y, int z){
661        if(u == v) return;
662        if(u == x) swap(u, v);
663        if(v == y) swap(v, z);
664        if(x == y) swap(x, z);
665
666        edge e1 = adj[u][v];
667        edge e2 = adj[v][z];
668        edge e3 = adj[z][y];
669        edge e4 = adj[y][x];
670
671        adj[u][v] = {v, z, -w};
672        adj[v][z] = {z, y, -w};
673        adj[z][y] = {y, x, -w};
674        adj[y][x] = {x, u, -w};
675
676    }
677
678    inline void flip(int u, int v, int w, int x, int y, int z){
679        if(u == v) return;
680        if(u == x) swap(u, v);
681        if(v == y) swap(v, z);
682        if(x == y) swap(x, z);
683
684        edge e1 = adj[u][v];
685        edge e2 = adj[v][z];
686        edge e3 = adj[z][y];
687        edge e4 = adj[y][x];
688
689        adj[u][v] = {v, z, -w};
690        adj[v][z] = {z, y, -w};
691        adj[z][y] = {y, x, -w};
692        adj[y][x] = {x, u, -w};
693
694    }
695
696    inline void flip(int u, int v, int w, int x, int y, int z){
697        if(u == v) return;
698        if(u == x) swap(u, v);
699        if(v == y) swap(v, z);
700        if(x == y) swap(x, z);
701
702        edge e1 = adj[u][v];
703        edge e2 = adj[v][z];
704        edge e3 = adj[z][y];
705        edge e4 = adj[y][x];
706
707        adj[u][v] = {v, z, -w};
708        adj[v][z] = {z, y, -w};
709        adj[z][y] = {y, x, -w};
710        adj[y][x] = {x, u, -w};
711
712    }
713
714    inline void flip(int u, int v, int w, int x, int y, int z){
715        if(u == v) return;
716        if(u == x) swap(u, v);
717        if(v == y) swap(v, z);
718        if(x == y) swap(x, z);
719
720        edge e1 = adj[u][v];
721        edge e2 = adj[v][z];
722        edge e3 = adj[z][y];
723        edge e4 = adj[y][x];
724
725        adj[u][v] = {v, z, -w};
726        adj[v][z] = {z, y, -w};
727        adj[z][y] = {y, x, -w};
728        adj[y][x] = {x, u, -w};
729
730    }
731
732    inline void flip(int u, int v, int w, int x, int y, int z){
733        if(u == v) return;
734        if(u == x) swap(u, v);
735        if(v == y) swap(v, z);
736        if(x == y) swap(x, z);
737
738        edge e1 = adj[u][v];
739        edge e2 = adj[v][z];
740        edge e3 = adj[z][y];
741        edge e4 = adj[y][x];
742
743        adj[u][v] = {v, z, -w};
744        adj[v][z] = {z, y, -w};
745        adj[z][y] = {y, x, -w};
746        adj[y][x] = {x, u, -w};
747
748    }
749
750    inline void flip(int u, int v, int w, int x, int y, int z){
751        if(u == v) return;
752        if(u == x) swap(u, v);
753        if(v == y) swap(v, z);
754        if(x == y) swap(x, z);
755
756        edge e1 = adj[u][v];
757        edge e2 = adj[v][z];
758        edge e3 = adj[z][y];
759        edge e4 = adj[y][x];
760
761        adj[u][v] = {v, z, -w};
762        adj[v][z] = {z, y, -w};
763        adj[z][y] = {y, x, -w};
764        adj[y][x] = {x, u, -w};
765
766    }
767
768    inline void flip(int u, int v, int w, int x, int y, int z){
769        if(u == v) return;
770        if(u == x) swap(u, v);
771        if(v == y) swap(v, z);
772        if(x == y) swap(x, z);
773
774        edge e1 = adj[u][v];
775        edge e2 = adj[v][z];
776        edge e3 = adj[z][y];
777        edge e4 = adj[y][x];
778
779        adj[u][v] = {v, z, -w};
780        adj[v][z] = {z, y, -w};
781        adj[z][y] = {y, x, -w};
782        adj[y][x] = {x, u, -w};
783
784    }
785
786    inline void flip(int u, int v, int w, int x, int y, int z){
787        if(u == v) return;
788        if(u == x) swap(u, v);
789        if(v == y) swap(v, z);
790        if(x == y) swap(x, z);
791
792        edge e1 = adj[u][v];
793        edge e2 = adj[v][z];
794        edge e3 = adj[z][y];
795        edge e4 = adj[y][x];
796
797        adj[u][v] = {v, z, -w};
798        adj[v][z] = {z, y, -w};
799        adj[z][y] = {y, x, -w};
800        adj[y][x] = {x, u, -w};
801
802    }
803
804    inline void flip(int u, int v, int w, int x, int y, int z){
805        if(u == v) return;
806        if(u == x) swap(u, v);
807        if(v == y) swap(v, z);
808        if(x == y) swap(x, z);
809
810        edge e1 = adj[u][v];
811        edge e2 = adj[v][z];
812        edge e3 = adj[z][y];
813        edge e4 = adj[y][x];
814
815        adj[u][v] = {v, z, -w};
816        adj[v][z] = {z, y, -w};
817        adj[z][y] = {y, x, -w};
818        adj[y][x] = {x, u, -w};
819
820    }
821
822    inline void flip(int u, int v, int w, int x, int y, int z){
823        if(u == v) return;
824        if(u == x) swap(u, v);
825        if(v == y) swap(v, z);
826        if(x == y) swap(x, z);
827
828        edge e1 = adj[u][v];
829        edge e2 = adj[v][z];
830        edge e3 = adj[z][y];
831        edge e4 = adj[y][x];
832
833        adj[u][v] = {v, z, -w};
834        adj[v][z] = {z, y, -w};
835        adj[z][y] = {y, x, -w};
836        adj[y][x] = {x, u, -w};
837
838    }
839
840    inline void flip(int u, int v, int w, int x, int y, int z){
841        if(u == v) return;
842        if(u == x) swap(u, v);
843        if(v == y) swap(v, z);
844        if(x == y) swap(x, z);
845
846        edge e1 = adj[u][v];
847        edge e2 = adj[v][z];
848        edge e3 = adj[z][y];
849        edge e4 = adj[y][x];
850
851        adj[u][v] = {v, z, -w};
852        adj[v][z] = {z, y, -w};
853        adj[z][y] = {y, x, -w};
854        adj[y][x] = {x, u, -w};
855
856    }
857
858    inline void flip(int u, int v, int w, int x, int y, int z){
859        if(u == v) return;
860        if(u == x) swap(u, v);
861       
```

```

70 }
71 inline int get_lca(int u, int v){
72     static int t=0;
73     for(++; u || v; swap(u, v)){
74         if(u == 0) continue;
75         if(vis[u] == t) return u;
76         vis[u] = t;
77         u = st[match[u]];
78         if(u) u = st[pa[u]];
79     }
80     return 0;
81 }
82 inline void add_blossom(int u, int lca, int v){
83     int b = n + 1;
84     while(b <= n_x && st[b]) ++b;
85     if(b > n_x) ++n_x;
86     lab[b] = 0, S[b] = 0;
87     match[b] = match[lca];
88     flower[b].clear();
89     flower[b].push_back(lca);
90     for(int x = u, y; x != lca; x = st[pa[y]]) {
91         flower[b].push_back(x),
92         flower[b].push_back(y = st[match[x]]),
93         q_push(y);
94     }
95     reverse(flower[b].begin() + 1,
96             flower[b].end());
97     for(int x = v, y; x != lca; x = st[pa[y]]) {
98         flower[b].push_back(x),
99         flower[b].push_back(y = st[match[x]]),
100        q_push(y);
101    }
102    set_st(b, b);
103    for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x]
104        ↪ [b].w = 0;
105    for(int x = 1; x <= n; ++x) flower_from[b][x] =
106        ↪ 0;
107    for(size_t i = 0 ; i < flower[b].size(); ++i){
108        int xs = flower[b][i];
109        for(int x = 1; x <= n_x; ++x)
110            if(g[b][x].w == 0 || e_delta(g(xs)[x])
111                ↪ < e_delta(g[b][x]))
112                g[b][x] = g[xs][x], g[x][b] = g[x]
113                    ↪ [xs];
114        for(int x = 1; x <= n; ++x)
115            if(flower_from[xs][x]) flower_from[b]
116                ↪ [x] = xs;
117    }
118    set_slack(b);
119 }
120 inline void expand_blossom(int b){ // S[b] == 1
121     for(size_t i = 0; i < flower[b].size(); ++i)
122         set_st(flower[b][i], flower[b][i]);
123     int xr = flower_from[b][g[b][pa[b]].u], pr =
124         ↪ get_pr(b, xr);
125     for(int i = 0; i < pr; i += 2){
126         int xs = flower[b][i], xns = flower[b][i +
127             ↪ 1];
128         pa[xs] = g[xns][xs].u;
129         S[xs] = 1, S[xns] = 0;
130         slack[xs] = 0, set_slack(xns);
131         q_push(xns);
132     }
133     S[xr] = 1, pa[xr] = pa[b];
134     for(size_t i = pr + 1; i < flower[b].size(); +
135         ↪ +i){
136         int xs = flower[b][i];
137         S[xs] = -1, set_slack(xs);
138     }

```

```

30     st[b] = 0;
31 }
32 inline bool on_found_edge(const edge &e){
33     int u = st[e.u], v = st[e.v];
34     if(S[v] == -1){
35         pa[v] = e.u, S[v] = 1;
36         int nu = st[match[v]];
37         slack[v] = slack[nu] = 0;
38         S[nu] = 0, q.push(nu);
39     }else if(S[v] == 0){
40         int lca = get_lca(u, v);
41         if(!lca) return augment(u, v), augment(v,
42             ↪ u), true;
43         else add_blossom(u, lca, v);
44     }
45     return false;
46 }
47 inline bool matching(){
48     memset(S + 1, -1, sizeof(int) * n_x);
49     memset(slack + 1, 0, sizeof(int) * n_x);
50     q = queue<int>();
51     for(int x = 1; x ≤ n_x; ++x)
52         if(st[x] == x && !match[x]) pa[x] = 0,
53             ↪ S[x] = 0, q.push(x);
54     if(q.empty()) return false;
55     for(;;){
56         while(q.size()){
57             int u = q.front(); q.pop();
58             if(S[st[u]] == 1) continue;
59             for(int v = 1; v ≤ n; ++v)
60                 if(g[u][v].w > 0 && st[u] != st[v])
61                     ↪ {
62                         if(e_delta(g[u][v]) == 0){
63                             if(on_found_edge(g[u]
64                                 ↪ [v])) return true;
65                         }else update_slack(u, st[v]);
66                     }
67             }
68             int d = INF;
69             for(int b = n + 1; b ≤ n_x; ++b)
70                 if(st[b] == b && S[b] == 1) d = min(d,
71                     ↪ lab[b]/2);
72             for(int x = 1; x ≤ n_x; ++x)
73                 if(st[x] == x && slack[x]){
74                     if(S[x] == -1) d = min(d,
75                         ↪ e_delta(g[slack[x]][x]));
76                     else if(S[x] == 0) d = min(d,
77                         ↪ e_delta(g[slack[x]][x])/2);
78                 }
79             for(int u = 1; u ≤ n; ++u){
80                 if(S[st[u]] == 0){
81                     if(lab[u] ≤ d) return 0;
82                     lab[u] -= d;
83                 }else if(S[st[u]] == 1) lab[u] += d;
84             }
85             for(int b = n+1; b ≤ n_x; ++b)
86                 if(st[b] == b){
87                     if(S[st[b]] == 0) lab[b] += d * 2;
88                     else if(S[st[b]] == 1) lab[b] -= d
89                         ↪ * 2;
90                 }
91             q = queue<int>();
92             for(int x = 1; x ≤ n_x; ++x)
93                 if(st[x] == x && slack[x] &&
94                     ↪ st[slack[x]] != x &&
95                     ↪ e_delta(g[slack[x]][x]) == 0)
96                     if(on_found_edge(g[slack[x]
97                         ↪ [x])) return true;
98             }
99         }
100     }
101 }
```

```

186     |     for(int b = n + 1; b <= n_x; ++b)
187     |     |     if(st[b] == b && S[b] == 1 && lab[b] ==
188     |     |     |     ~0)expand_blossom(b);
189     |
190     |     return false;
191 }
192 inline pair<long long, int> solve(){
193     memset(match + 1, 0, sizeof(int) * n);
194     n_x = n;
195     int n_matches = 0;
196     long long tot_weight = 0;
197     for(int u = 0; u <= n; ++u) st[u] = u,
198         |     flower[u].clear();
199     int w_max = 0;
200     for(int u = 1; u <= n; ++u)
201         |     for(int v = 1; v <= n; ++v){
202             |             flower_from[u][v] = (u == v ? u : 0);
203             |             w_max = max(w_max, g[u][v].w);
204         }
205     for(int u = 1; u <= n; ++u) lab[u] = w_max;
206     while(matching()) ++n_matches;
207     for(int u = 1; u <= n; ++u)
208         |     if(match[u] && match[u] < u)
209             |             tot_weight += g[u][match[u]].w;
210     return make_pair(tot_weight, n_matches);
211 }
212 inline void init(){
213     for(int u = 1; u <= n; ++u)
214         |     for(int v = 1; v <= n; ++v)
215             |             g[u][v]=edge(u, v, 0);
216 }
217 };

```

3.6.4 原理

设图 G 的Tutte矩阵是 \tilde{A} , 首先是最基础的引理:

- G 的最大匹配大小是 $\frac{1}{2}\text{rank } \tilde{A}$.
- $(\tilde{A}^{-1})_{i,j} \neq 0$ 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.

3.7 支配树

记得建反图!

```

1 vector<int> G[maxn], R[maxn], son[maxn]; // R是反图,
2     |     → son存的是sdom树上的儿子
3
4 int ufs[maxn];
5
6 int idom[maxn], sdom[maxn], anc[maxn]; // anc:
    |     → sdom的dfn最小的祖先

```

```

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
50     son[sdom[x]].push_back(x);
51     ufs[x] = p[x];
52
53     for (int u : son[p[x]]) {
54         findufs(u);
55         idom[u] = (sdom[u] == sdom[anc[u]] ? p[x] :
56                     anc[u]);
57     }
58
59     son[p[x]].clear();
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 }

```

3.8 2-SAT

如果限制满足对称性(每个命题的逆否命题对应的边也存在), 那么可以使用Tarjan算法求SCC搞定.

具体来说就是，如果某个变量的两个点在同一SCC中则显然无解，否则按拓扑序倒序尝试选择每个SCC即可。

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

```

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

如果要字典序最小就用DFS，注意可以压位优化。另外代码是0-base的。

```

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

```

10 memset(last, -1, sizeof(last));
11
12 void AddEdge(int x, int y, int z) {
13     e[len].to = y;
14     e[len].cap = z;
15     e[len].prev = last[x];
16     last[x] = len++;
17 }
18
19 void addedge(int x, int y, int z) {
20     AddEdge(x, y, z);
21     AddEdge(y, x, 0);
22 }
23
24 void bfs() {
25     int head = 0, tail = 0;
26     memset(d, -1, sizeof(int) * (t + 5));
27     q[tail++] = s;
28     d[s] = 0;
29
30     while (head != tail) {
31         int x = q[head++];
32         for (int i = last[x]; ~i; i = e[i].prev)
33             if (e[i].cap > 0 && d[e[i].to] == -1) {
34                 d[e[i].to] = d[x] + 1;
35                 q[tail++] = e[i].to;
36             }
37     }
38 }
39
40 int dfs(int x, int a) {
41     if (x == t || !a)
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))) > 0) {
48             e[i].cap -= f;
49             e[i ^ 1].cap += f;
50             flow += f;
51             a -= f;
52
53             if (!a)
54                 break;
55         }
56
57     return flow;
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 }
```

3.9 最大流

3.9.1 Dinic

```

1 // 注意Dinic适用于二分图或分层图，对于一般稀疏图ISAP更
2 // 优，稠密图则HLPP更优
3
4 struct edge {
5     int to, cap, prev;
6 } e[maxn * 2];
7
8 int last[maxn], len, d[maxn], cur[maxn], q[maxn];
9
// main函数里要初始化
```

3.9.2 ISAP

可能有毒，慎用。

```

1 // 注意ISAP适用于一般稀疏图，对于二分图或分层图情
2 // 况Dinic比较优，稠密图则HLPP更优
3
4 // 边的定义
// 这里没有记录起点和反向边，因为反向边即为正向边xor 1,
// → 起点即为反向边的终点
```

```

5 struct edge {
6     int to, cap, prev;
7 } e[maxe * 2];
8
9 // 全局变量和数组定义
10 int last[maxn], cnte = 0, d[maxn], p[maxn], c[maxn],
11    ~cur[maxn], q[maxn];
12 int n, m, s, t; // s, t一定要开成全局变量
13
14 void AddEdge(int x, int y, int z) {
15     e[cnte].to = y;
16     e[cnte].cap = z;
17     e[cnte].prev = last[x];
18     last[x] = cnte++;
19 }
20
21 void addedge(int x, int y, int z) {
22     AddEdge(x, y, z);
23     AddEdge(y, x, 0);
24 }
25
26 // 预处理到t的距离标号
27 // 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
28 void bfs() {
29     memset(d, -1, sizeof(d));
30
31     int head = 0, tail = 0;
32     d[t] = 0;
33     q[tail++] = t;
34
35     while (head != tail) {
36         int x = q[head++];
37
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
75             flow += augment();
76             x = s;
77         }
78
79         bool ok = false;
80         for (int &i = cur[x]; ~i; i = e[i].prev)
81             if (e[i].cap && d[x] == d[e[i].to] + 1) {
82                 p[e[i].to] = i;
83                 x = e[i].to;
84             }
85
86         if (!ok) { // 修改距离标号
87             int tmp = n - 1;
88             for (int i = last[x]; ~i; i = e[i].prev)
89                 if (e[i].cap)
90                     tmp = min(tmp, d[e[i].to] + 1);
91
92             if (!--c[d[x]])
93                 break; // gap优化, 一定要加上
94
95             c[d[x]] = tmp++;
96             cur[x] = last[x];
97
98             if (x != s)
99                 x = e[p[x] ^ 1].to;
100
101        }
102    }
103
104 // 重要! main函数最前面一定要加上如下初始化
105 memset(last, -1, sizeof(last));
106

```

```

72         flow += augment();
73         x = s;
74     }
75
76     bool ok = false;
77     for (int &i = cur[x]; ~i; i = e[i].prev)
78         if (e[i].cap && d[x] == d[e[i].to] + 1) {
79             p[e[i].to] = i;
80             x = e[i].to;
81         }
82         ok = true;
83     break;
84 }
85
86 if (!ok) { // 修改距离标号
87     int tmp = n - 1;
88     for (int i = last[x]; ~i; i = e[i].prev)
89         if (e[i].cap)
90             tmp = min(tmp, d[e[i].to] + 1);
91
92     if (!--c[d[x]])
93         break; // gap优化, 一定要加上
94
95     c[d[x]] = tmp++;
96     cur[x] = last[x];
97
98     if (x != s)
99         x = e[p[x] ^ 1].to;
100
101 }
102 return flow;
103
104 // 重要! main函数最前面一定要加上如下初始化
105 memset(last, -1, sizeof(last));
106

```

3.9.3 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[maxe * 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     return h[s] < 2 * n;
52 }
53 void push(int x) {
54     for (int i = last[x]; ~i; i = e[i].prev)
55         if (e[i].cap && h[x] == h[e[i].to] + 1) {
56             int d = min(ex[x], (long long)e[i].cap);
57
58             e[i].cap -= d;
59             e[i ^ 1].cap += d;
60             ex[x] -= d;
61             ex[e[i].to] += d;
62
63             if (e[i].to != s && e[i].to != t &&
64                 !inq[e[i].to]) {
65                 heap.push(e[i].to);
66                 inq[e[i].to] = true;
67             }
68
69             if (!ex[x])
70                 break;
71 }
72 void relabel(int x) {
73     h[x] = 2 * n;
74
75     for (int i = last[x]; ~i; i = e[i].prev)
76         if (e[i].cap)
77             h[x] = min(h[x], h[e[i].to] + 1);
78 }
79
80 long long hlpp() {
81     if (!bfs())
82         return 0;
83
84     // memset(gap, 0, sizeof(int) * 2 * n);
85     h[s] = n;
86
87     for (int i = 1; i <= n; i++)
88         gap[h[i]]++;
89
90     for (int i = last[s]; ~i; i = e[i].prev)
91         if (e[i].cap) {
92             int d = e[i].cap;
93
94             e[i].cap -= d;
95             e[i ^ 1].cap += d;
96             ex[s] -= d;
97             ex[e[i].to] += d;
98

```

```

99
100    if (e[i].to != s && e[i].to != t &&
101        !inq[e[i].to]) {
102        heap.push(e[i].to);
103        inq[e[i].to] = true;
104    }
105
106    while (!heap.empty()) {
107        int x = heap.top();
108        heap.pop();
109        inq[x] = false;
110
111        push(x);
112        if (ex[x]) {
113            if (!--gap[h[x]]) { // gap
114                for (int i = 1; i <= n; i++)
115                    if (i != s && i != t && h[i] >
116                        h[x])
117                        h[i] = n + 1;
118
119            relabel(x);
120            ++gap[h[x]];
121            heap.push(x);
122            inq[x] = true;
123        }
124
125    }
126
127    return ex[t];
128
129 //记得初始化
130 memset(last, -1, sizeof(last));

```

3.10 费用流

3.10.1 SPFA费用流

```

1 constexpr int maxn = 20005, maxm = 200005;
2
3 struct edge {
4     int to, prev, cap, w;
5 } e[maxm * 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
13     memset(d, -63, sizeof(d));
14     memset(p, -1, sizeof(p));
15
16     queue<int> q;
17
18     q.push(s);
19     d[s] = 0;
20
21     while (!q.empty()) {
22         int x = q.front();
23         q.pop();
24         inq[x] = false;
25
26         for (int i = last[x]; ~i; i = e[i].prev)
27             if (e[i].cap) {
28                 int y = e[i].to;
29
30                 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 int mcmf(int s, int t) {
41     int ans = 0;
42
43     while (spfa(s), d[t] > 0) {
44         int flow = 0x3f3f3f3f;
45         for (int x = t; x != s; x = e[p[x] ^ 1].to)
46             flow = min(flow, e[p[x]].cap);
47
48         ans += flow * d[t];
49
50         for (int x = t; x != s; x = e[p[x] ^ 1].to) {
51             e[p[x]].cap -= flow;
52             e[p[x] ^ 1].cap += flow;
53         }
54     }
55
56     return ans;
57 }

58 void add(int x, int y, int c, int w) {
59     e[cnte].to = y;
60     e[cnte].cap = c;
61     e[cnte].w = w;
62
63     e[cnte].prev = last[x];
64     last[x] = cnte++;
65 }

66 void addedge(int x, int y, int c, int w) {
67     add(x, y, c, w);
68     add(y, x, 0, -w);
69 }

70 }
```

3.10.2 Dijkstra费用流

有的地方也叫原始-对偶费用流。

原理和求多源最短路的Johnson算法是一样的，都是给每个点维护一个势 h_u ，使得对任何有向边 $u \rightarrow v$ 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 void addedge(int x, int y, int z, int w) {
21     Adde(x, y, z, w);
22     Adde(y, x, 0, -w);
23 }

24 }

25 void dijkstra() {
26     memset(d, 63, sizeof(d));
27     memset(vis, 0, sizeof(vis));
28
29     priority_queue<pair<long long, int>> heap;
30
31     d[s] = 0;
32     heap.push(make_pair(0ll, s));
33
34     while (!heap.empty()) {
35         int x = heap.top().second;
36         heap.pop();
37
38         if (vis[x])
39             continue;
40
41         vis[x] = true;
42         for (int i = last[x]; ~i; i = e[i].prev)
43             if (e[i].cap > 0 && d[e[i].to] > d[x] +
44                 e[i].w + h[x] - h[e[i].to]) {
45                 d[e[i].to] = d[x] + e[i].w + h[x] -
46                 h[e[i].to];
47                 p[e[i].to] = i;
48                 heap.push(make_pair(-d[e[i].to],
49                                     e[i].to));
50             }
51     }
52 }
```

```

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

```

81     return make_pair(flow, cost);
82 }
83
84 // 记得初始化
85 memset(last, -1, sizeof(last));

```

3.10.3 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 // Runs in O(<max_flow> * log(V * max_edge_cost)) = O(
3 //   ↪ V^3 * log(V * C))
4 // Really fast in practice, 3e4 edges are fine.
5 // Operates on integers, costs are multiplied by N!!
6
7 #include <bits/stdc++.h>
8 using namespace std;
9
10 // source: unknown
11 template<typename flow_t = int, typename cost_t = int>
12 struct mcSFlow {
13     struct Edge {
14         cost_t c;
15         flow_t f;
16         int to, rev;
17         Edge(int _to, cost_t _c, flow_t _f, int _rev):
18             ↪ c(_c), f(_f), to(_to), rev(_rev) {}
19     };
20
21     static constexpr cost_t INF_COST =
22         ↪ numeric_limits<cost_t>::max() / 2;
23
24     cost_t eps;
25     int N, S, T;
26     vector<vector<Edge>> G;
27     vector<unsigned int> isq, cur;
28     vector<flow_t> ex;
29     vector<cost_t> h;
30
31     mcSFlow(int _N, int _S, int _T): eps(0), N(_N),
32             ↪ S(_S), T(_T), G(_N) {}
33
34     void add_edge(int a, int b, flow_t cap, cost_t
35             ↪ cost) {
36         assert(cap >= 0);
37         assert(a >= 0 && a < N && b >= 0 && b < N);
38
39         if (a == b) {
40             assert(cost >= 0);
41             return;
42         }
43
44         cost *= N;
45         eps = max(eps, abs(cost));
46         G[a].emplace_back(b, cost, cap, G[b].size());
47         G[b].emplace_back(a, -cost, 0, G[a].size() -
48             ↪ 1);
49     }
50
51     void add_flow(Edge &e, flow_t f) {
52         Edge &back = G[e.to][e.rev];
53
54         if (!ex[e.to] && f)
55             hs[h[e.to]].push_back(e.to);
56
57         e.f -= f;
58         ex[e.to] += f;
59         back.f += f;

```

```

54     ex[back.to] -= f;
55 }
56
57 vector<vector<int>> hs;
58 vector<int> co;
59
60 flow_t max_flow() {
61     ex.assign(N, 0);
62     h.assign(N, 0);
63     hs.resize(2 * N);
64     co.assign(2 * N, 0);
65     cur.assign(N, 0);
66     h[S] = N;
67     ex[T] = 1;
68     co[0] = N - 1;
69
70     for (auto &e : G[S])
71         add_flow(e, e.f);
72
73     if (hs[0].size())
74         for (int hi = 0; hi >= 0;) {
75             int u = hs[hi].back();
76             hs[hi].pop_back();
77
78             while (ex[u] > 0) { // discharge u
79                 if (cur[u] == G[u].size())
80                     h[u] = 1e9;
81
82                 for (unsigned int i = 0; i <
83                     ↪ G[u].size(); ++i) {
84                     auto &e = G[u][i];
85
86                     if (e.f && h[u] > h[e.to] +
87                         ↪ 1) {
88                         h[u] = h[e.to] + 1,
89                         ↪ cur[u] = i;
90
91                     if (++co[h[u]], !--co[hi] && hi
92                         ↪ < N)
93                         for (int i = 0; i < N; ++i)
94                             if (hi < h[i] && h[i] <
95                                 ↪ N) {
96                                 --co[h[i]];
97                                 h[i] = N + 1;
98
99                         hi = h[u];
100
101                     else if (G[u][cur[u]].f && h[u] ==
102                         ↪ h[G[u][cur[u]].to] + 1)
103                         add_flow(G[u][cur[u]],
104                             ↪ min(ex[u], G[u]
105                                 ↪ [cur[u]].f));
106
107                     else
108                         ++cur[u];
109
110                     while (hi >= 0 && hs[hi].empty())
111                         --hi;
112
113                 return -ex[S];
114
115             void push(Edge &e, flow_t amt) {
116                 if (e.f < amt)
117                     amt = e.f;

```

```

115     e.f -= amt;
116     ex[e.to] += amt;
117     G[e.to][e.rev].f += amt;
118     ex[G[e.to][e.rev].to] -= amt;
119 }
120
121 void relabel(int vertex) {
122     cost_t newHeight = -INFCOST;
123
124     for (unsigned int i = 0; i < G[vertex].size(); ++i) {
125         Edge const &e = G[vertex][i];
126
127         if (e.f && newHeight < h[e.to] - e.c) {
128             newHeight = h[e.to] - e.c;
129             cur[vertex] = i;
130         }
131     }
132
133     h[vertex] = newHeight - eps;
134 }
135
136 static constexpr int scale = 2;
137
138 pair<flow_t, cost_t> minCostMaxFlow() {
139     cost_t retCost = 0;
140
141     for (int i = 0; i < N; ++i)
142         for (Edge &e : G[i])
143             retCost += e.c * (e.f);
144
145
146     //find max-flow
147     flow_t retFlow = max_flow();
148     h.assign(N, 0);
149     ex.assign(N, 0);
150     isq.assign(N, 0);
151     cur.assign(N, 0);
152     queue<int> q;
153
154     for (; eps; eps >= scale) {
155         //refine
156         fill(cur.begin(), cur.end(), 0);
157
158         for (int i = 0; i < N; ++i)
159             for (auto &e : G[i])
160                 if (h[i] + e.c - h[e.to] < 0 &&
161                     ~e.f)
162                     push(e, e.f);
163
164         for (int i = 0; i < N; ++i) {
165             if (ex[i] > 0) {
166                 q.push(i);
167                 isq[i] = 1;
168             }
169
170         // make flow feasible
171         while (!q.empty()) {
172             int u = q.front();
173             q.pop();
174             isq[u] = 0;
175
176             while (ex[u] > 0) {
177                 if (cur[u] == G[u].size())
178                     relabel(u);

```

```

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             if (ex[u] == 0)
198                 break;
199         }
200
201         if (eps > 1 && eps >> scale == 0)
202             eps = 1 << scale;
203
204         for (int i = 0; i < N; ++i)
205             for (Edge &e : G[i])
206                 retCost -= e.c * (e.f);
207
208
209         return make_pair(retFlow, retCost / 2 / N);
210
211     }
212
213     flow_t getFlow(Edge const &e) {
214         return G[e.to][e.rev].f;
215     }
216
217     int main() {
218
219         int n, m;
220         scanf("%d%d", &n, &m);
221
222         mcSFlow<long long, long long> mcmf(n, 0, n - 1);
223
224         while (m--) {
225             int x, y, z, w;
226             scanf("%d%d%d%d", &x, &y, &z, &w);
227
228             mcmf.add_edge(x - 1, y - 1, z, w);
229
230         auto [flow, cost] = mcmf.minCostMaxFlow();
231
232         printf("%lld %lld\n", flow, cost);
233
234     }

```

3.11 网络流原理

3.11.1 最大流

- 判断一条边是否必定满流

在残量网络中跑一遍Tarjan, 如果某条满流边的两端处于同一SCC中则说明它不一定满流. (因为可以找出包含反向边的环, 增广之后就不满流了.)

3.11.2 最小割

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。

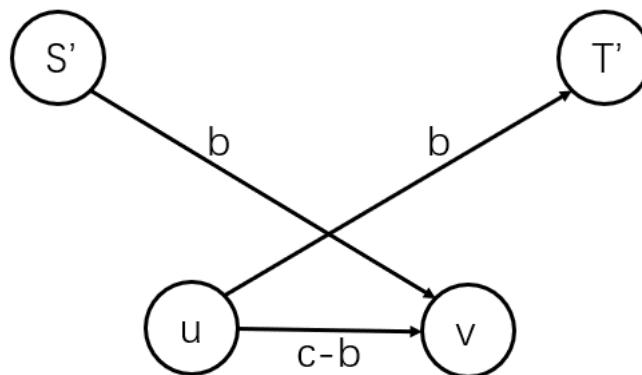
如果一条边是可行边，我们就需要把它删掉，同时进行退流， $u \rightarrow s$ 和 $t \rightarrow v$ 都退掉等同于这条边容量的流量。

退流用Dinic实现即可。

3.11.3 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     memset(last, -1, sizeof(last));
5
6     int n, m, src, sink;
7     scanf("%d%d%d%d", &n, &m, &src, &sink);
8     s = n + 1;
9     t = n + 2;
10
11    while (m--) {
12        int x, y, b, c;
13        scanf("%d%d%d%d", &x, &y, &b, &c);
14
15        addedge(x, y, c - b);
16
17        ex[y] += b;
18    }
  
```

```

19     ex[x] -= b;
20
21 }
22
23 for (int i = 1; i <= n; i++) {
24     id[i] = cnte;
25
26     if (ex[i] >= 0)
27         addedge(s, i, ex[i]);
28     else
29         addedge(i, t, -ex[i]);
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
48 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。

3.11.4 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。

3.11.5 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。

3.12 Prufer序列

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

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

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

Prufer编码的性质

- 每个至少2个结点的树都唯一对应一个Prufer编码。（当然也就可做无根树哈希。）
- 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     }
37     return code;
38 }
```

线性重构树代码：

```

1 // 0-based
2 vector<pair<int, int>> pruefer_decode(vector<int> const
3     &code) {
4     int n = code.size() + 2;
5     vector<int> degree(n, 1);
6     for (int i : code) degree[i]++;
7
8     int ptr = 0;
9     while (degree[ptr] != 1) ptr++;
10    int leaf = ptr;
```

```

10
11     vector<pair<int, int>> edges;
12     for (int v : code) {
13         edges.emplace_back(leaf, v);
14         if (--degree[v] == 1 && v < ptr)
15             leaf = v;
16         else {
17             ptr++;
18             while (degree[ptr] != 1)
19                 ptr++;
20             leaf = ptr;
21         }
22     }
23     edges.emplace_back(leaf, n - 1);
24     return edges;
25 }
```

3.13 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

3.14 其他

3.14.1 Stoer-Wagner全局最小割

```

1 const int N = 601;
2 int fa[N], siz[N], edge[N][N];
3
4 int find(int x) {
5     return fa[x] == x ? x : fa[x] = find(fa[x]);
6 }
7
8 int dist[N], vis[N], bin[N];
9 int n, m;
10
11 int contract(int& s, int& t) { // Find s, t
12     memset(dist, 0, sizeof(dist));
13     memset(vis, false, sizeof(vis));
14
15     int i, j, k, mincut, maxc;
16
17     for (i = 1; i <= n; i++) {
18         k = -1;
19         maxc = -1;
20
21         for (j = 1; j <= n; j++)
22             if (!bin[j] && !vis[j] && dist[j] > maxc) {
23                 k = j;
24                 maxc = dist[j];
25             }
26
27         if (k == -1)
28             return mincut;
29
30         s = t;
31         t = k;
```

```
32     mincut = maxc;
33     vis[k] = true;
34
35     for (j = 1; j <= n; j++)
36     |     if (!bin[j] && !vis[j]) dist[j] += edge[k]
37     |         ↪ [j];
38
39     return mincut;
40 }
41
42 const int inf = 0x3f3f3f3f;
43
44 int Stoer_Wagner() {
45     int mincut, i, j, s, t, ans;
46     for (mincut = inf, i = 1; i < n; i++) {
47         ans = contract(s, t);
48         bin[t] = true;
49
50         if (mincut > ans)
51             mincut = ans;
52         if (mincut == 0)
53             return 0;
54
55         for (j = 1; j <= n; j++)
56         |         if (!bin[j])
57         |             edge[s][j] = (edge[j][s] += edge[j]
58         |                 ↪ [t]);
59
60     return mincut;
61 }
62
63 int main() {
64     cin >> n >> m;
65
66     if (m < n - 1) {
67         cout << 0;
68         return 0;
69     }
70
71     for (int i = 1; i <= n; ++i)
72         fa[i] = i, siz[i] = 1;
73
74     for (int i = 1, u, v, w; i <= m; ++i) {
75         cin >> u >> v >> w;
76
77         int fu = find(u), fv = find(v);
78         if (fu != fv) {
79             if (siz[fu] > siz[fv]) swap(fu, fv);
80             fa[fu] = fv, siz[fv] += siz[fu];
81         }
82
83         edge[u][v] += w, edge[v][u] += w;
84     }
85
86     int fr = find(1);
87
88     if (siz[fr] != n) {
89         cout << 0;
90         return 0;
91     }
92
93     cout << Stoer_Wagner();
94
95     return 0;
96 }
```

4 数据结构

4.1 线段树

4.1.1 非递归线段树

- 如果 $M = 2^k$, 则只能维护 $[1, M - 2]$ 范围
- 找叶子: i 对应的叶子就是 $i + M$
- 单点修改: 找到叶子然后向上跳
- 区间查询: 左右区间各扩展一位, 转换成开区间查询

```

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

区间修改要标记永久化, 并且求区间和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         }
8     }
9 }
```

```

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

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 void modify(int l, int r, int &o) {
14     if (!o)
15         o = ++seg_cnt;
16
17     if (s <= l && t >= r) {
```

```

18     cover[o] += d;
19     refresh(l, r, o);

20     return;
21 }

22 int mid = (l + r) / 2;

23 if (s <= mid)
24     modify(l, mid, lc[o]);
25 if (t > mid)
26     modify(mid + 1, r, rc[o]);

27 refresh(l, r, o);
28 }

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

```

4.1.3 历史和

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

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

“既然标记会复合 就说明在两个标记中间没有经过任何 pushup 操作

也就是说一个这两个标记对应着 相同的 0 的数量 以及 相同的 1 的数量

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

```

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

```

```

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

```

136 }

4.2 陈丹琦分治

4.2.1 动态图连通性(分治并查集)

```

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

```

```

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

```

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
36    ↳ 序, 所以要复制一份
37
38    CDQ2(l, r);
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);
69
70         t[k++] = b[i++];
71     }
72
73     while (j <= r) {
74         if (!b[j].ins)
75             ans += query(b[j].z - 1);
76
77         t[k++] = b[j++];
78     }
79
80     for (i = l; i <= mid; i++)
81         if (b[i].ins)
82             add(b[i].z, -1);
83
84     copy(t + l, t + r + 1, b + l);
85 }

```

4.3 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:
2   ↳ query
3 long long qk[maxn]; // 修改和询问可以一起存
4
5 int ans[maxn];
6
7 void solve(int l, int r, vector<int> v) { // 如果想卡常
8   ↳ 可以用数组, 然后只需要传一个数组的l, r; 递归的时候类
9   ↳ 似归并反过来, 开两个辅助数组, 处理完再复制回去即可
10    if (v.empty())
11        return;
12
13    if (l == r) {
14        for (int i : v)
15            if (op[i] == 2)
16                ans[i] = l;
17
18        return;
19    }
20
21    int mid = (l + r) / 2;
22
23    vector<int> vl, vr;
24
25    for (int i : v) {
26        if (op[i] == 1) {
27            if (qk[i] <= mid)
28
```

```

25     |     vl.push_back(i);
26     | else {
27     |     update(ql[i], qr[i], 1); // update是区间
28     |     ↪ 加
29     |     vr.push_back(i);
30     | }
31     | else {
32     |     long long tmp = query(ql[i], qr[i]);
33
34     |     if (qk[i] ≤ tmp) // 因为是k大数查询
35     |         vr.push_back(i);
36     |     else {
37     |         qk[i] -= tmp;
38     |         vl.push_back(i);
39     |     }
40   }
41
42   for (int i : vr)
43     if (op[i] == 1)
44       update(ql[i], qr[i], -1);
45
46   v.clear();
47
48   solve(l, mid, vl);
49   solve(mid + 1, r, vr);
50 }
51
52
53 int main() {
54   int n, m;
55   scanf("%d%d", &n, &m);
56
57   M = 1;
58   while (M < n + 2)
59     M *= 2;
60
61   for (int i = 1; i ≤ m; i++)
62     scanf("%d%d%d%lld", &op[i], &ql[i], &qr[i],
63           ↪ &qk[i]);
64
65   vector<int> v;
66   for (int i = 1; i ≤ m; i++)
67     v.push_back(i);
68
69   solve(1, 1e9, v);
70
71   for (int i = 1; i ≤ m; i++)
72     if (op[i] == 2)
73       printf("%d\n", ans[i]);
74
75   return 0;
}

```

4.4 平衡树

pb_ds平衡树参见8.11.Public Based DataStructure(PB_DS)
(90页).

4.4.1 Treap

```

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

```

```

9   void refresh() {
10     size = ch[0] → size + ch[1] → size + 1;
11   } // 更新子树大小(和附加信息, 如果有的话)
12 } null[maxn], *root = null, *ptr = null; // 数组名叫
13 ↪ 做null是为了方便开哨兵节点
14 // 如果需要删除而空间不能直接开下所有结点, 则需要再写一个
15 ↪ 垃圾回收
16 // 注意: 数组里的元素一定不能delete, 否则会导致RE
17 // 重要! 在主函数最开始一定要加上以下预处理:
18 null → ch[0] = null → ch[1] = null;
19 null → size = 0;
20
21 // 伪构造函数 O(1)
22 // 为了方便, 在结点类外面再定义一个伪构造函数
23 node *newnode(int x) { // 键值为x
24   ***ptr = node(x);
25   ptr → ch[0] = ptr → ch[1] = null;
26   return ptr;
27 }
28
29 // 插入键值 期望O(log n)
30 // 需要调用旋转
31 void insert(int x, node *&rt) { // rt为当前结点, 建议调
32   → 用时传入root, 下同
33   if (rt == null) {
34     rt = newnode(x);
35     return;
36   }
37
38   int d = x > rt → key;
39   insert(x, rt → ch[d]);
40   rt → refresh();
41
42   if (rt → ch[d] → p < rt → p)
43     rot(rt, d ^ 1);
44
45 // 删除一个键值 期望O(log n)
46 // 要求键值必须存在至少一个, 否则会导致RE
47 // 需要调用旋转
48 void erase(int x, node *&rt) {
49   if (x == rt → key) {
50     if (rt → ch[0] != null && rt → ch[1] != null)
51       → {
52         int d = rt → ch[0] → p < rt → ch[1] →
53           ↪ p;
54         rot(rt, d);
55         erase(x, rt → ch[d]);
56       }
57     else
58       rt = rt → ch[rt → ch[0] == null];
59   }
60   else
61     erase(x, rt → ch[x > rt → key]);
62
63   if (rt != null)
64     rt → refresh();
65
66 // 求元素的排名(严格小于键值的个数 + 1) 期望O(log n)
67 // 非递归
68 int rank(int x, node *rt) {
69   int ans = 1, d;
70   while (rt != null) {
71     if ((d = x > rt → key))
72       ans += rt → ch[0] → size + 1;
73
74     rt = rt → ch[d];
75   }
76
77   return ans;
78 }

```

```

73 }
74
75     return ans;
76 }
77
78 // 返回排名第k(从1开始)的键值对应的指针 期望O(log n)
79 // 非递归
80 node *kth(int x, node *rt) {
81     int d;
82     while (rt != null) {
83         if (x == rt → ch[0] → size + 1)
84             return rt;
85
86         if ((d = x > rt → ch[0] → size))
87             x -= rt → ch[0] → size + 1;
88
89         rt = rt → ch[d];
90     }
91
92     return rt;
93 }
94
95 // 返回前驱(最大的比给定键值小的键值)对应的指针 期
96 // →望O(log n)
97 // 非递归
98 node *pred(int x, node *rt) {
99     node *y = null;
100    int d;
101
102    while (rt != null) {
103        if ((d = x > rt → key))
104            y = rt;
105
106        rt = rt → ch[d];
107    }
108
109    return y;
110 }
111
112 // 返回后继(最小的比给定键值大的键值)对应的指针 期
113 // →望O(log n)
114 // 非递归
115 node *succ(int x, node *rt) {
116     node *y = null;
117     int d;
118
119     while (rt != null) {
120         if ((d = x < rt → key))
121             y = rt;
122
123         rt = rt → ch[d ^ 1];
124     }
125
126     return y;
127 }
128
129 // 旋转(Treap版本) O(1)
130 // 平衡树基础操作
131 // 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
132 void rot(node *&x, int d) { // x为被转下去的结点, 会被修
133     // 改以维护树结构
134     node *y = x → ch[d ^ 1];
135
136     x → ch[d ^ 1] = y → ch[d];
137     y → ch[d] = x;
138
139     x → refresh();
140     (x = y) → refresh();
141 }

```

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 refresh() {
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(y);
27         z → ch[0] = merge(x, y → ch[0]);
28     else {
29         z = copied(x);
30         z → ch[1] = merge(x → ch[1], y);
31     }
32
33     z → refresh(); // 因为每次只有一边会递归到儿子, 所
34     // →以z不可能取到null
35     return z;
36 }
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 → refresh();
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 → refresh();
60         pi.first = y;
61     }
62
63     return pi;
64 }

```

```

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

```

4.4.3 Splay

如果插入的话可以直接找到底然后splay一下，也可以直接splay前驱后继。

```

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

```

```

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

```

4.5 树链剖分

4.5.1 动态树形DP(最大权独立集)

```

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

```

```

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

```

```

149     |     |     son[x] = y;
150
151     |     sz[x] += sz[y];
152 }
153
154 void dfs2(int x) {
155     if (x == son[p[x]]) {
156         top[x] = top[p[x]];
157     } else {
158         top[x] = x;
159
160         for (int y : G[x])
161             if (y != p[x])
162                 dfs2(y);
163
164         dp[x][1] = w[x];
165         for (int y : G[x])
166             if (y != p[x] && y != son[x]) {
167                 dp[x][1] += dptr[y][0];
168                 dp[x][0] += max(dptr[y][0], dptr[y][1]);
169             }
170
171         if (top[x] == x) {
172             tw.clear();
173
174             for (int u = x; u; u = son[u])
175                 tw.push_back(make_pair(dp[u][0], dp[u]
176                                         → [1]));
177
178             seg[x].build((int)tw.size());
179
180             tie(dptr[x][0], dptr[x][1]) = seg[x].getpair();
181
182             heap.push(seg[x].getval());
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;

```

```

214     scanf("%d%d", &n, &m);
215
216     for (int i = 1; i ≤ n; i++)
217         scanf("%d", &w[i]);
218
219     for (int i = 1; i < n; i++) {
220         int x, y;
221         scanf("%d%d", &x, &y);
222
223         G[x].push_back(y);
224         G[y].push_back(x);
225     }
226
227     dfs1(1);
228     dfs2(1);
229
230     while (m--) {
231         int x, dat;
232         scanf("%d%d", &x, &dat);
233
234         modify(x, dat);
235
236         printf("%d\n", heap.top());
237     }
238
239     return 0;
240 }

```

4.6 树分治

4.6.1 动态树分治

```

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

```

```

32 // 找重心 O(n)
33 int getcenter(int x, int s) {
34     int head = 0, tail = 0;
35     q[tail++] = x;
36
37     while (head != tail) {
38         x = q[head++];
39         size[x] = 1; // 这里不需要清空，因为以后要用的话
39             ↪ 一定会重新赋值
40         son[x] = 0;
41
42         for (int i = 0; i < (int)G[x].size(); i++) {
43             if (!vis[G[x][i]] && G[x][i] != p[x]) {
44                 p[G[x][i]] = x;
45                 q[tail++] = G[x][i];
46             }
47         }
48
49         for (int i = tail - 1; i; i--) {
50             x = q[i];
51             size[p[x]] += size[x];
52
53             if (size[x] > size[son[p[x]]])
54                 son[p[x]] = x;
55         }
56
57         x = q[0];
58         while (son[x] && size[son[x]] * 2 ≥ s)
59             x = son[x];
60
61     return x;
62 }

```

```

64 // 预处理距离 O(n)
65 // 方便起见，这里直接用了笨一点的方法，O(n log n)全存下来
66 void getdis(int x, int k, int rt) {
67     int head = 0, tail = 0;
68     q[tail++] = x;
69
70     while (head != tail) {
71         x = q[head++];
72         size[x] = 1;
73         id[x][k] = rt;
74
75         for (int i = 0; i < (int)G[x].size(); i++) {
76             if (!vis[G[x][i]] && G[x][i] != p[x]) {
77                 p[G[x][i]] = x;
78                 d[G[x][i]][k] = d[x][k] + W[x][i];
79
80                 q[tail++] = G[x][i];
81             }
82         }
83
84         for (int i = tail - 1; i; i--)
85             size[p[q[i]]] += size[q[i]]; // 后面递归建树要用
85             ↪ 到子问题大小
86     }
87
88 // 修改 O(log n)
89 void modify(int x) {
90     if (col[x])
91         ca[x]--;
92     else
93         ca[x]++; // 记得先特判自己作为重心的那层
94
95     for (int u = p[x], k = depth[x] - 1; u; u = p[u],
95         ↪ k--) {
96         if (col[x])
97             a[u] -= d[x][k];
98             ca[u]--;

```

```

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

```

4.6.2 紫荆花之恋

稍微重构了一下，修改了代码风格。

另外这个是BFS版本，跑得比DFS要快不少。（虽然主要复杂度并不在重构上）

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 100005, maxk = 49;
6 constexpr double alpha = .75;
7
8 mt19937 rnd(23333333);
9
10 struct node {
11     int key, size, p;
12     node *ch[2];
13
14     node() {}
15
16     node(int key) : key(key), size(1), p(rnd()) {}
17
18     inline void update() {
19         size = ch[0] → size + ch[1] → size + 1;
20     }
21 } null[maxn * maxk], *pt = null;
22
23 vector<node*> pool;
24
25 node *newnode(int val) {
26     node *x;
27
28     if (!pool.empty()) {
29         x = pool.back();
30         pool.pop_back();
31     } else
32         x = ++pt;
33
34     *x = node(val);
35

```

```

36     x → ch[0] = x → ch[1] = null;
37
38     return x;
39 }
40
41 void rot(node *&x, int d) {
42     node *y = x → ch[d ^ 1];
43     x → ch[d ^ 1] = y → ch[d];
44     y → ch[d] = x;
45
46     x → update();
47     (x = y) → update();
48 }
49
50 void insert(node *&o, int x) {
51     if (o == null) {
52         o = newnode(x);
53         return;
54     }
55
56     int d = (x > o → key);
57
58     insert(o → ch[d], x);
59     o → update();
60
61     if (o → ch[d] → p < o → p)
62         rot(o, d ^ 1);
63 }
64
65 int get_order(node *o, int x) {
66     int ans = 0;
67
68     while (o != null) {
69         int d = (x > o → key);
70
71         if (d)
72             ans += o → ch[0] → size + 1;
73
74         o = o → ch[d];
75     }
76
77     return ans;
78 }
79
80 void destroy(node *x) {
81     if (x == null)
82         return;
83
84     pool.push_back(x);
85     destroy(x → ch[0]);
86     destroy(x → ch[1]);
87 }
88
89 struct my_tree { // 封装了一下，如果不卡内存直接换成PBDS就好了
90     node *rt;
91
92     my_tree() : rt(null) {}
93
94     void clear() {
95         ::destroy(rt);
96         rt = null;
97     }
98
99     void insert(int x) {
100         ::insert(rt, x);
101     }
102
103     int order_of_key(int x) { // less than x
104         return ::get_order(rt, x);
105     }

```

```

105 }
106 } tr[maxn], tre[maxn][maxk];
107
108 vector<pair<int, int>> G[maxn];
109
110 int p[maxn], depth[maxn], d[maxn][maxk], rid[maxn]
111    ↪ [maxk];
112 int sz[maxn], siz[maxn][maxk], q[maxn];
113 bool vis[maxn];
114
115 int w[maxn];
116
117 void destroy(int o) {
118     int head = 0, tail = 0;
119     q[tail++] = o;
120     vis[o] = false;
121
122     while (head != tail) {
123         int x = q[head++];
124         tr[x].clear();
125
126         for (int i = depth[o]; i ≤ depth[x]; i++) {
127             tre[x][i].clear();
128             d[x][i] = rid[x][i] = siz[x][i] = 0;
129         }
130
131         for (auto pi : G[x]) {
132             int y = pi.first;
133
134             if (vis[y] && depth[y] ≥ depth[o]) {
135                 vis[y] = false;
136                 q[tail++] = y;
137             }
138         }
139     }
140 }
141
142 int getcenter(int o, int s) {
143     int head = 0, tail = 0;
144     q[tail++] = o;
145
146     while (head != tail) {
147         int x = q[head++];
148         sz[x] = 1;
149
150         for (auto pi : G[x]) {
151             int y = pi.first;
152
153             if (!vis[y] && y != p[x]) {
154                 p[y] = x;
155                 q[tail++] = y;
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         }
176     }
177 }

```

```

172     }
173 
174     if (!ok)
175     | break;
176 
177 }
178 
179 return x;
180 }

181 void getdis(int st, int o, int k) {
182     int head = 0, tail = 0;
183     q[tail++] = st;
184 
185     while (head != tail) {
186         int x = q[head++];
187         sz[x] = 1;
188         rid[x][k] = st;
189 
190         tr[o].insert(d[x][k] - w[x]);
191         tre[st][k].insert(d[x][k] - w[x]);
192 
193         for (auto pi : G[x]) {
194             int y = pi.first, val = pi.second;
195 
196             if (!vis[y] && y != p[x]) {
197                 p[y] = x;
198                 d[y][k] = d[x][k] + val;
199                 q[tail++] = y;
200             }
201         }
202     }
203 
204     for (int i = tail - 1; i; i--)
205         sz[p[q[i]]] += sz[q[i]];
206 
207     siz[st][k] = sz[st];
208 }

209 }

210 void rebuild(int x, int s, int pr) {
211     x = getcenter(x, s);
212     vis[x] = true;
213     p[x] = pr;
214     depth[x] = depth[pr] + 1;
215     sz[x] = s;
216 
217     tr[x].insert(-w[x]);
218 
219     for (auto pi : G[x]) {
220         int y = pi.first, val = pi.second;
221 
222         if (!vis[y]) {
223             p[y] = x;
224             d[y][depth[x]] = val;
225             getdis(y, x, depth[x]);
226         }
227     }
228 
229     for (auto pi : G[x]) {
230         int y = pi.first;
231 
232         if (!vis[y])
233             rebuild(y, sz[y], x);
234     }
235 }

236 long long add_node(int x, int nw) { // nw是边权
237     depth[x] = depth[p[x]] + 1;
238     sz[x] = 1;
239 }

240 }

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 return tmp;
273 }

274 int main() {
275     null → ch[0] = null → ch[1] = null;
276     null → size = 0;
277 
278     int n;
279     scanf("%*d%d", &n);
280 
281     scanf("%*d%*d%d", &w[1]);
282     vis[1] = true;
283     sz[1] = 1;
284     tr[1].insert(-w[1]);
285 
286     printf("0\n");
287 
288     long long ans = 0;
289 
290     for (int i = 2; i ≤ n; i++) {
291         int nw;
292         scanf("%d%d%d", &p[i], &nw, &w[i]);
293 
294         p[i] ≈ (ans % 1000000000);
295 
296         G[i].push_back(make_pair(p[i], nw));
297         G[p[i]].push_back(make_pair(i, nw));
298 
299         ans += add_node(i, nw);
300 
301         printf("%lld\n", ans);
302     }
303 
304     return 0;
305 }

```

4.7 LCT动态树

4.7.1 不换根(弹飞绵羊)

```

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

```

```

64 // 需要调用旋转
65 void splay(node *x) {
66     while (!isroot(x)) {
67         if (isroot(x → p)) {
68             rot(x → p, dir(x) ^ 1);
69             break;
70         }
71
72         if (dir(x) == dir(x → p))
73             rot(x → p → p, dir(x → p) ^ 1);
74         else
75             rot(x → p, dir(x) ^ 1);
76         rot(x → p, dir(x) ^ 1);
77     }
78 }
79
80 // 旋转(LCT版本) O(1)
81 // 平衡树基本操作
82 // 要求对应儿子必须存在, 否则会导致后续各种莫名其妙的问题
83 void rot(node *x, int d) {
84     node *y = x → ch[d ^ 1];
85
86     y → p = x → p;
87     if (!isroot(x))
88         x → p → ch[dir(x)] = y;
89
90     if ((x → ch[d ^ 1] = y → ch[d]) != null)
91         y → ch[d] → p = x;
92     (y → ch[d] = x) → p = y;
93
94     x → refresh();
95     y → refresh();
96 }

```

4.7.2 换根/维护生成树

```

1 #define isroot(x) ((x) → p == null || ((x) → p →
2     → ch[0] != (x) && (x) → p → ch[1] != (x)))
3 #define dir(x) ((x) == (x) → p → ch[1])
4
5 using namespace std;
6
7 const int maxn = 200005;
8
9 struct node{
10     int key, mx, pos;
11     bool rev;
12     node *ch[2], *p;
13
14     node(int key = 0): key(key), mx(key), pos(-1),
15         → rev(false) {}
16
17     void pushdown() {
18         if (!rev)
19             return;
20
21         ch[0] → rev ^= true;
22         ch[1] → rev ^= true;
23         swap(ch[0], ch[1]);
24
25         if (pos != -1)
26             pos ^= 1;
27
28         rev = false;
29     }
30
31     void refresh() {
32         mx = key;
33         pos = -1;
34     }
35 }
36
37
38
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

```

```

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

```

4.7.3 维护子树信息

```

1 // 这个东西虽然只需要抄板子但还是极其难写，常数极其巨大，  

2 // →没必要的时候就不要用  

3 // 如果维护子树最小值就需要套一个可删除的堆来维护，复杂度  

4 // →会变成O(n log^2 n)  

5 // 注意由于这道题与边权有关，需要边权拆点变点权  

6 // 宏定义  

7 #define isroot(x) ((x) → p == null || ((x) != (x) → p  

8 // → ch[0] && (x) != (x) → p → ch[1]))  

9 #define dir(x) ((x) == (x) → p → ch[1])  

10 // 节点类定义  

11 struct node { // 以维护子树中黑点到根距离和为例  

12     int w, chain_cnt, tree_cnt;  

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

14     // →子树反转，需要维护两个方向的信息  

15     bool rev, col;  

16     node *ch[2], *p;  

17  

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

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

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

21  

22     inline void pushdown() {  

23         if(!rev)  

24             return;  

25  

26         ch[0]→rev ≈ true;  

27         ch[1]→rev ≈ true;  

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

29         swap(suml, sumr);  

30  

31         rev = false;  

32     }  

33  

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

35         // →就pushdown一下  

36         // pushdown();  

37  

38         sum = ch[0] → sum + ch[1] → sum + w;  

39         suml = (ch[0] → rev ? ch[0] → sumr : ch[0] →  

40         // → suml) + (ch[1] → rev ? ch[1] → sumr :  

41         // → ch[1] → suml) + (tree_cnt + ch[1] →  

42         // → chain_cnt) * (ch[0] → sum + w) + tree_sum;  

43         sumr = (ch[0] → rev ? ch[0] → suml : ch[0] →  

44         // → sumr) + (ch[1] → rev ? ch[1] → suml :  

45         // → ch[1] → sumr) + (tree_cnt + ch[0] →  

46         // → chain_cnt) * (ch[1] → sum + w) + tree_sum;  

47         chain_cnt = ch[0] → chain_cnt + ch[1] →  

48         // → chain_cnt + tree_cnt;  

49     }  

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

51  

52     // 封装构造函数  

53     node *newnode(int w) {  

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

55         // →个队列维护可用结点  

56         nodes.pop();  

57         initialize(x);  

58         x → w = w;  

59         x → refresh();  

60         return x;  

61     }  

62  

63     // 封装初始化函数  

64     // 记得在进行操作之前对所有结点调用一遍  

65     inline void initialize(node *x) {  

66         *x = node();  

67         x → ch[0] = x → ch[1] = x → p = null;  

68  

69         // 注意一下在Access的同时更新子树信息的方法  

70         node *access(node *x) {  

71             node *y = null;  

72  

73             while (x != null) {  

74                 splay(x);  

75  

76                 x → tree_cnt += x → ch[1] → chain_cnt - y →  

77                 // → chain_cnt;  

78                 x → tree_sum += (x → ch[1] → rev ? x →  

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

80                 // → suml;  

81                 x → ch[1] = y;  

82  

83                 (y = x) → refresh();  

84                 x = x → p;  

85             }  

86  

87             return y;  

88         }  

89  

90         // 找到一个点所在连通块的根  

91         // 对比原版没有变化  

92         node *getroot(node *x) {  

93             x = access(x);  

94  

95             while (x → pushdown(), x → ch[0] != null)  

96                 x = x → ch[0];  

97             splay(x);  

98  

99             return x;  

100        }  

101  

102        // 换根，同样没有变化  

103        void makeroot(node *x) {  

104            access(x);  

105            splay(x);  

106            x → rev ≈ true;  

107            x → pushdown();  

108  

109            // 连接两个点  

110            // !!! 注意这里必须把两者都变成根，因为只能修改根结点  

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

112                makeroot(x);  

113                makeroot(y);  

114  

115                x → p = y;  

116                y → tree_cnt += x → chain_cnt;  

117                y → tree_sum += x → suml;  

118                y → refresh();  

119  

120                // 删除一条边  

121                // 对比原版没有变化  

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

123                    makeroot(x);  

124                    access(y);  

125                    splay(y);  

126  

127                    y → ch[0] → p = null;  

128                    y → ch[0] = null;  

129                    y → refresh();  

130  

131                }  

132  

133                // 修改/询问一个点，这里以询问为例  

134                // 如果是修改就在换根之后搞一些操作

```

```

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

```

```

19     void push(long long x) {
20         if (x > (-INF) >> 2)
21             q1.push(x);
22     }
23
24     void erase(long long x) {
25         if (x > (-INF) >> 2)
26             q2.push(x);
27     }
28
29     long long top() {
30         if (empty())
31             return -INF;
32
33         while (!q2.empty() && q1.top() == q2.top())
34             q1.pop();
35         q2.pop();
36
37         return q1.top();
38     }
39
40     long long top2() {
41         if (size() < 2)
42             return -INF;
43
44         long long a = top();
45         erase(a);
46         long long b = top();
47         push(a);
48         return a + b;
49     }
50
51     int size() {
52         return q1.size() - q2.size();
53     }
54
55     bool empty() {
56         return q1.size() == q2.size();
57     }
58 } heap; // 全局堆维护每条链的最大子段和
59
60 struct node {
61     long long sum, maxsum, prefix, suffix;
62     int key;
63     binary_heap heap; // 每个点的堆存的是它的子树中到它的
64     // 最远距离, 如果它是黑点的话还会包括自己
65     node *ch[2], *p;
66     bool rev;
67     node(int k = 0): sum(k), maxsum(-INF),
68         prefix(-INF), suffix(-INF), key(k), rev(false) {}
69     inline void pushdown() {
70         if (!rev)
71             return;
72
73         ch[0] -> rev = true;
74         ch[1] -> rev = true;
75         swap(ch[0], ch[1]);
76         swap(prefix, suffix);
77         rev = false;
78     }
79     inline void refresh() {
80         pushdown();
81         ch[0] -> pushdown();
82         ch[1] -> pushdown();
83         sum = ch[0] -> sum + ch[1] -> sum + key;
84         prefix = max(ch[0] -> prefix,
85             ch[0] -> sum + key + ch[1] ->
86             prefix);
87         suffix = max(ch[1] -> suffix,
88             ch[1] -> sum + key + ch[0] ->
89             suffix);
90     }
91 }
92
93 constexpr int maxn = 100005;
94 constexpr long long INF = 1000000000000000000ll;
95
96 struct binary_heap {
97     __gnu_pbds::priority_queue<long long, less<long
98     long>, binary_heap_tag> q1, q2;
99     binary_heap() {}
100 }
```

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 }
```

```

87         ch[1] → sum + key + ch[0] →
88             ↪ suffix);
89     maxsum = max(max(ch[0] → maxsum, ch[1] →
90                     ↪ maxsum),
91                 ch[0] → suffix + key + ch[1] →
92                     ↪ prefix);
93
94     if (!heap.empty()) {
95         prefix = max(prefix,
96                       ch[0] → sum + key +
97                           ↪ heap.top());
98         suffix = max(suffix,
99                       ch[1] → sum + key +
100                          ↪ heap.top());
101        maxsum = max(maxsum, max(ch[0] → suffix,
102                           ch[1] → prefix) +
103                               ↪ key +
104                                 ↪ heap.top());
105
106        if (heap.size() > 1) {
107            maxsum = max(maxsum, heap.top2() +
108                           ↪ key);
109        }
110    }
111 }
112 null[maxn << 1], *ptr = null;
113
114 void addedge(int, int, int);
115 void deledge(int, int);
116 void modify(int, int, int);
117 void modify_color(int);
118 node *newnode(int);
119 node *access(node *);
120 void makeroot(node *);
121 void link(node *, node *);
122 void cut(node *, node *);
123 void splay(node *);
124 void rot(node *, int);
125
126 queue<node *> freenodes;
127 tree<pair<int, int>, node *> mp;
128
129 bool col[maxn] = {false};
130 char c;
131 int n, m, k, x, y, z;
132
133 int main() {
134     null → ch[0] = null → ch[1] = null → p = null;
135     scanf("%d%d%d", &n, &m, &k);
136
137     for (int i = 1; i ≤ n; i++)
138         newnode(0);
139
140     heap.push(0);
141
142     while (k--) {
143         scanf("%d", &x);
144
145         col[x] = true;
146         null[x].heap.push(0);
147     }
148
149     for (int i = 1; i < n; i++) {
150         scanf("%d%d%d", &x, &y, &z);
151
152         if (x > y)
153             swap(x, y);
154         addedge(x, y, z);
155     }
156
157     while (m--) {
158         scanf("%c%d", &c, &x);
159
160         if (c == 'A') {
161             scanf("%d", &y);
162
163             if (x > y)
164                 swap(x, y);
165             deledge(x, y);
166         }
167         else if (c == 'B') {
168             scanf("%d%d", &y, &z);
169
170             if (x > y)
171                 swap(x, y);
172             addedge(x, y, z);
173         }
174         else if (c == 'C') {
175             scanf("%d%d", &y, &z);
176
177             if (x > y)
178                 swap(x, y);
179             modify(x, y, z);
180         }
181         else
182             modify_color(x);
183
184         printf("%lld\n", (heap.top() > 0 ? heap.top() :
185                           ↪ -1));
186     }
187
188     return 0;
189 }
190
191 void addedge(int x, int y, int z) {
192     node *tmp;
193
194     if (freenodes.empty())
195         tmp = newnode(z);
196     else {
197         tmp = freenodes.front();
198         freenodes.pop();
199         *tmp = node(z);
200     }
201
202     tmp → ch[0] = tmp → ch[1] = tmp → p = null;
203
204     heap.push(tmp → maxsum);
205     link(tmp, null + x);
206     link(tmp, null + y);
207     mp[make_pair(x, y)] = tmp;
208
209 }
210
211 void deledge(int x, int y) {
212     node *tmp = mp[make_pair(x, y)];
213
214     cut(tmp, null + x);
215     cut(tmp, null + y);
216
217     freenodes.push(tmp);
218     heap.erase(tmp → maxsum);
219     mp.erase(make_pair(x, y));
220
221 }
222
223 void modify(int x, int y, int z) {
224     node *tmp = mp[make_pair(x, y)];
225
226     makeroot(tmp);
227     tmp → pushdown();
228
229     heap.erase(tmp → maxsum);
230     tmp → key = z;
231     tmp → refresh();
232     heap.push(tmp → maxsum);
233
234 }
235
236 void rot(node *x, int y);
237
238 void refresh(node *x);
239
240 void pushdown(node *x);
241
242 void pushup(node *x);
243
244 void makeroot(node *x);
245
246 void link(node *x, node *y);
247
248 void cut(node *x, node *y);
249
250 void swap(int &x, int &y);
251
252 void deledge(node *x, node *y);
253
254 void modify_color(int x);
255
256 void modify(int x, int y, int z);
257
258 void addedge(int x, int y, int z);
259
260 void splay(node *x);
261
262 void pushup(node *x);
263
264 void pushdown(node *x);
265
266 void refresh(node *x);
267
268 void makeroot(node *x);
269
270 void link(node *x, node *y);
271
272 void cut(node *x, node *y);
273
274 void swap(int &x, int &y);
275
276 void deledge(node *x, node *y);
277
278 void modify_color(int x);
279
280 void modify(int x, int y, int z);
281
282 void addedge(int x, int y, int z);
283
284 void splay(node *x);
285
286 void pushup(node *x);
287
288 void pushdown(node *x);
289
290 void refresh(node *x);
291
292 void makeroot(node *x);
293
294 void link(node *x, node *y);
295
296 void cut(node *x, node *y);
297
298 void swap(int &x, int &y);
299
300 void deledge(node *x, node *y);
301
302 void modify_color(int x);
303
304 void modify(int x, int y, int z);
305
306 void addedge(int x, int y, int z);
307
308 void splay(node *x);
309
310 void pushup(node *x);
311
312 void pushdown(node *x);
313
314 void refresh(node *x);
315
316 void makeroot(node *x);
317
318 void link(node *x, node *y);
319
320 void cut(node *x, node *y);
321
322 void swap(int &x, int &y);
323
324 void deledge(node *x, node *y);
325
326 void modify_color(int x);
327
328 void modify(int x, int y, int z);
329
330 void addedge(int x, int y, int z);
331
332 void splay(node *x);
333
334 void pushup(node *x);
335
336 void pushdown(node *x);
337
338 void refresh(node *x);
339
340 void makeroot(node *x);
341
342 void link(node *x, node *y);
343
344 void cut(node *x, node *y);
345
346 void swap(int &x, int &y);
347
348 void deledge(node *x, node *y);
349
350 void modify_color(int x);
351
352 void modify(int x, int y, int z);
353
354 void addedge(int x, int y, int z);
355
356 void splay(node *x);
357
358 void pushup(node *x);
359
360 void pushdown(node *x);
361
362 void refresh(node *x);
363
364 void makeroot(node *x);
365
366 void link(node *x, node *y);
367
368 void cut(node *x, node *y);
369
370 void swap(int &x, int &y);
371
372 void deledge(node *x, node *y);
373
374 void modify_color(int x);
375
376 void modify(int x, int y, int z);
377
378 void addedge(int x, int y, int z);
379
380 void splay(node *x);
381
382 void pushup(node *x);
383
384 void pushdown(node *x);
385
386 void refresh(node *x);
387
388 void makeroot(node *x);
389
390 void link(node *x, node *y);
391
392 void cut(node *x, node *y);
393
394 void swap(int &x, int &y);
395
396 void deledge(node *x, node *y);
397
398 void modify_color(int x);
399
400 void modify(int x, int y, int z);
401
402 void addedge(int x, int y, int z);
403
404 void splay(node *x);
405
406 void pushup(node *x);
407
408 void pushdown(node *x);
409
410 void refresh(node *x);
411
412 void makeroot(node *x);
413
414 void link(node *x, node *y);
415
416 void cut(node *x, node *y);
417
418 void swap(int &x, int &y);
419
420 void deledge(node *x, node *y);
421
422 void modify_color(int x);
423
424 void modify(int x, int y, int z);
425
426 void addedge(int x, int y, int z);
427
428 void splay(node *x);
429
430 void pushup(node *x);
431
432 void pushdown(node *x);
433
434 void refresh(node *x);
435
436 void makeroot(node *x);
437
438 void link(node *x, node *y);
439
440 void cut(node *x, node *y);
441
442 void swap(int &x, int &y);
443
444 void deledge(node *x, node *y);
445
446 void modify_color(int x);
447
448 void modify(int x, int y, int z);
449
450 void addedge(int x, int y, int z);
451
452 void splay(node *x);
453
454 void pushup(node *x);
455
456 void pushdown(node *x);
457
458 void refresh(node *x);
459
460 void makeroot(node *x);
461
462 void link(node *x, node *y);
463
464 void cut(node *x, node *y);
465
466 void swap(int &x, int &y);
467
468 void deledge(node *x, node *y);
469
470 void modify_color(int x);
471
472 void modify(int x, int y, int z);
473
474 void addedge(int x, int y, int z);
475
476 void splay(node *x);
477
478 void pushup(node *x);
479
480 void pushdown(node *x);
481
482 void refresh(node *x);
483
484 void makeroot(node *x);
485
486 void link(node *x, node *y);
487
488 void cut(node *x, node *y);
489
490 void swap(int &x, int &y);
491
492 void deledge(node *x, node *y);
493
494 void modify_color(int x);
495
496 void modify(int x, int y, int z);
497
498 void addedge(int x, int y, int z);
499
500 void splay(node *x);
501
502 void pushup(node *x);
503
504 void pushdown(node *x);
505
506 void refresh(node *x);
507
508 void makeroot(node *x);
509
510 void link(node *x, node *y);
511
512 void cut(node *x, node *y);
513
514 void swap(int &x, int &y);
515
516 void deledge(node *x, node *y);
517
518 void modify_color(int x);
519
520 void modify(int x, int y, int z);
521
522 void addedge(int x, int y, int z);
523
524 void splay(node *x);
525
526 void pushup(node *x);
527
528 void pushdown(node *x);
529
530 void refresh(node *x);
531
532 void makeroot(node *x);
533
534 void link(node *x, node *y);
535
536 void cut(node *x, node *y);
537
538 void swap(int &x, int &y);
539
540 void deledge(node *x, node *y);
541
542 void modify_color(int x);
543
544 void modify(int x, int y, int z);
545
546 void addedge(int x, int y, int z);
547
548 void splay(node *x);
549
550 void pushup(node *x);
551
552 void pushdown(node *x);
553
554 void refresh(node *x);
555
556 void makeroot(node *x);
557
558 void link(node *x, node *y);
559
560 void cut(node *x, node *y);
561
562 void swap(int &x, int &y);
563
564 void deledge(node *x, node *y);
565
566 void modify_color(int x);
567
568 void modify(int x, int y, int z);
569
570 void addedge(int x, int y, int z);
571
572 void splay(node *x);
573
574 void pushup(node *x);
575
576 void pushdown(node *x);
577
578 void refresh(node *x);
579
580 void makeroot(node *x);
581
582 void link(node *x, node *y);
583
584 void cut(node *x, node *y);
585
586 void swap(int &x, int &y);
587
588 void deledge(node *x, node *y);
589
590 void modify_color(int x);
591
592 void modify(int x, int y, int z);
593
594 void addedge(int x, int y, int z);
595
596 void splay(node *x);
597
598 void pushup(node *x);
599
600 void pushdown(node *x);
601
602 void refresh(node *x);
603
604 void makeroot(node *x);
605
606 void link(node *x, node *y);
607
608 void cut(node *x, node *y);
609
610 void swap(int &x, int &y);
611
612 void deledge(node *x, node *y);
613
614 void modify_color(int x);
615
616 void modify(int x, int y, int z);
617
618 void addedge(int x, int y, int z);
619
620 void splay(node *x);
621
622 void pushup(node *x);
623
624 void pushdown(node *x);
625
626 void refresh(node *x);
627
628 void makeroot(node *x);
629
630 void link(node *x, node *y);
631
632 void cut(node *x, node *y);
633
634 void swap(int &x, int &y);
635
636 void deledge(node *x, node *y);
637
638 void modify_color(int x);
639
640 void modify(int x, int y, int z);
641
642 void addedge(int x, int y, int z);
643
644 void splay(node *x);
645
646 void pushup(node *x);
647
648 void pushdown(node *x);
649
650 void refresh(node *x);
651
652 void makeroot(node *x);
653
654 void link(node *x, node *y);
655
656 void cut(node *x, node *y);
657
658 void swap(int &x, int &y);
659
660 void deledge(node *x, node *y);
661
662 void modify_color(int x);
663
664 void modify(int x, int y, int z);
665
666 void addedge(int x, int y, int z);
667
668 void splay(node *x);
669
670 void pushup(node *x);
671
672 void pushdown(node *x);
673
674 void refresh(node *x);
675
676 void makeroot(node *x);
677
678 void link(node *x, node *y);
679
680 void cut(node *x, node *y);
681
682 void swap(int &x, int &y);
683
684 void deledge(node *x, node *y);
685
686 void modify_color(int x);
687
688 void modify(int x, int y, int z);
689
690 void addedge(int x, int y, int z);
691
692 void splay(node *x);
693
694 void pushup(node *x);
695
696 void pushdown(node *x);
697
698 void refresh(node *x);
699
700 void makeroot(node *x);
701
702 void link(node *x, node *y);
703
704 void cut(node *x, node *y);
705
706 void swap(int &x, int &y);
707
708 void deledge(node *x, node *y);
709
710 void modify_color(int x);
711
712 void modify(int x, int y, int z);
713
714 void addedge(int x, int y, int z);
715
716 void splay(node *x);
717
718 void pushup(node *x);
719
720 void pushdown(node *x);
721
722 void refresh(node *x);
723
724 void makeroot(node *x);
725
726 void link(node *x, node *y);
727
728 void cut(node *x, node *y);
729
730 void swap(int &x, int &y);
731
732 void deledge(node *x, node *y);
733
734 void modify_color(int x);
735
736 void modify(int x, int y, int z);
737
738 void addedge(int x, int y, int z);
739
740 void splay(node *x);
741
742 void pushup(node *x);
743
744 void pushdown(node *x);
745
746 void refresh(node *x);
747
748 void makeroot(node *x);
749
750 void link(node *x, node *y);
751
752 void cut(node *x, node *y);
753
754 void swap(int &x, int &y);
755
756 void deledge(node *x, node *y);
757
758 void modify_color(int x);
759
760 void modify(int x, int y, int z);
761
762 void addedge(int x, int y, int z);
763
764 void splay(node *x);
765
766 void pushup(node *x);
767
768 void pushdown(node *x);
769
770 void refresh(node *x);
771
772 void makeroot(node *x);
773
774 void link(node *x, node *y);
775
776 void cut(node *x, node *y);
777
778 void swap(int &x, int &y);
779
780 void deledge(node *x, node *y);
781
782 void modify_color(int x);
783
784 void modify(int x, int y, int z);
785
786 void addedge(int x, int y, int z);
787
788 void splay(node *x);
789
790 void pushup(node *x);
791
792 void pushdown(node *x);
793
794 void refresh(node *x);
795
796 void makeroot(node *x);
797
798 void link(node *x, node *y);
799
800 void cut(node *x, node *y);
801
802 void swap(int &x, int &y);
803
804 void deledge(node *x, node *y);
805
806 void modify_color(int x);
807
808 void modify(int x, int y, int z);
809
810 void addedge(int x, int y, int z);
811
812 void splay(node *x);
813
814 void pushup(node *x);
815
816 void pushdown(node *x);
817
818 void refresh(node *x);
819
820 void makeroot(node *x);
821
822 void link(node *x, node *y);
823
824 void cut(node *x, node *y);
825
826 void swap(int &x, int &y);
827
828 void deledge(node *x, node *y);
829
830 void modify_color(int x);
831
832 void modify(int x, int y, int z);
833
834 void addedge(int x, int y, int z);
835
836 void splay(node *x);
837
838 void pushup(node *x);
839
840 void pushdown(node *x);
841
842 void refresh(node *x);
843
844 void makeroot(node *x);
845
846 void link(node *x, node *y);
847
848 void cut(node *x, node *y);
849
850 void swap(int &x, int &y);
851
852 void deledge(node *x, node *y);
853
854 void modify_color(int x);
855
856 void modify(int x, int y, int z);
857
858 void addedge(int x, int y, int z);
859
860 void splay(node *x);
861
862 void pushup(node *x);
863
864 void pushdown(node *x);
865
866 void refresh(node *x);
867
868 void makeroot(node *x);
869
870 void link(node *x, node *y);
871
872 void cut(node *x, node *y);
873
874 void swap(int &x, int &y);
875
876 void deledge(node *x, node *y);
877
878 void modify_color(int x);
879
880 void modify(int x, int y, int z);
881
882 void addedge(int x, int y, int z);
883
884 void splay(node *x);
885
886 void pushup(node *x);
887
888 void pushdown(node *x);
889
890 void refresh(node *x);
891
892 void makeroot(node *x);
893
894 void link(node *x, node *y);
895
896 void cut(node *x, node *y);
897
898 void swap(int &x, int &y);
899
900 void deledge(node *x, node *y);
901
902 void modify_color(int x);
903
904 void modify(int x, int y, int z);
905
906 void addedge(int x, int y, int z);
907
908 void splay(node *x);
909
910 void pushup(node *x);
911
912 void pushdown(node *x);
913
914 void refresh(node *x);
915
916 void makeroot(node *x);
917
918 void link(node *x, node *y);
919
920 void cut(node *x, node *y);
921
922 void swap(int &x, int &y);
923
924 void deledge(node *x, node *y);
925
926 void modify_color(int x);
927
928 void modify(int x, int y, int z);
929
930 void addedge(int x, int y, int z);
931
932 void splay(node *x);
933
934 void pushup(node *x);
935
936 void pushdown(node *x);
937
938 void refresh(node *x);
939
940 void makeroot(node *x);
941
942 void link(node *x, node *y);
943
944 void cut(node *x, node *y);
945
946 void swap(int &x, int &y);
947
948 void deledge(node *x, node *y);
949
950 void modify_color(int x);
951
952 void modify(int x, int y, int z);
953
954 void addedge(int x, int y, int z);
955
956 void splay(node *x);
957
958 void pushup(node *x);
959
960 void pushdown(node *x);
961
962 void refresh(node *x);
963
964 void makeroot(node *x);
965
966 void link(node *x, node *y);
967
968 void cut(node *x, node *y);
969
970 void swap(int &x, int &y);
971
972 void deledge(node *x, node *y);
973
974 void modify_color(int x);
975
976 void modify(int x, int y, int z);
977
978 void addedge(int x, int y, int z);
979
980 void splay(node *x);
981
982 void pushup(node *x);
983
984 void pushdown(node *x);
985
986 void refresh(node *x);
987
988 void makeroot(node *x);
989
990 void link(node *x, node *y);
991
992 void cut(node *x, node *y);
993
994 void swap(int &x, int &y);
995
996 void deledge(node *x, node *y);
997
998 void modify_color(int x);
999
1000 void modify(int x, int y, int z);
1001
1002 void addedge(int x, int y, int z);
1003
1004 void splay(node *x);
1005
1006 void pushup(node *x);
1007
1008 void pushdown(node *x);
1009
1010 void refresh(node *x);
1011
1012 void makeroot(node *x);
1013
1014 void link(node *x, node *y);
1015
1016 void cut(node *x, node *y);
1017
1018 void swap(int &x, int &y);
1019
1020 void deledge(node *x, node *y);
1021
1022 void modify_color(int x);
1023
1024 void modify(int x, int y, int z);
1025
1026 void addedge(int x, int y, int z);
1027
1028 void splay(node *x);
1029
1030 void pushup(node *x);
1031
1032 void pushdown(node *x);
1033
1034 void refresh(node *x);
1035
1036 void makeroot(node *x);
1037
1038 void link(node *x, node *y);
1039
1040 void cut(node *x, node *y);
1041
1042 void swap(int &x, int &y);
1043
1044 void deledge(node *x, node *y);
1045
1046 void modify_color(int x);
1047
1048 void modify(int x, int y, int z);
1049
1050 void addedge(int x, int y, int z);
1051
1052 void splay(node *x);
1053
1054 void pushup(node *x);
1055
1056 void pushdown(node *x);
1057
1058 void refresh(node *x);
1059
1060 void makeroot(node *x);
1061
1062 void link(node *x, node *y);
1063
1064 void cut(node *x, node *y);
1065
1066 void swap(int &x, int &y);
1067
1068 void deledge(node *x, node *y);
1069
1070 void modify_color(int x);
1071
1072 void modify(int x, int y, int z);
1073
1074 void addedge(int x, int y, int z);
1075
1076 void splay(node *x);
1077
1078 void pushup(node *x);
1079
1080 void pushdown(node *x);
1081
1082 void refresh(node *x);
1083
1084 void makeroot(node *x);
1085
1086 void link(node *x, node *y);
1087
1088 void cut(node *x, node *y);
1089
1090 void swap(int &x, int &y);
1091
1092 void deledge(node *x, node *y);
1093
1094 void modify_color(int x);
1095
1096 void modify(int x, int y, int z);
1097
1098 void addedge(int x, int y, int z);
1099
1100 void splay(node *x);
1101
1102 void pushup(node *x);
1103
1104 void pushdown(node *x);
1105
1106 void refresh(node *x);
1107
1108 void makeroot(node *x);
1109
1110 void link(node *x, node *y);
1111
1112 void cut(node *x, node *y);
1113
1114 void swap(int &x, int &y);
1115
1116 void deledge(node *x, node *y);
1117
1118 void modify_color(int x);
1119
1120 void modify(int x, int y, int z);
1121
1122 void addedge(int x, int y, int z);
1123
1124 void splay(node *x);
1125
1126 void pushup(node *x);
1127
1128 void pushdown(node *x);
1129
1130 void refresh(node *x);
1131
1132 void makeroot(node *x);
1133
1134 void link(node *x, node *y);
1135
1136 void cut(node *x, node *y);
1137
1138 void swap(int &x, int &y);
1139
1140 void deledge(node *x, node *y);
1141
1142 void modify_color(int x);
1143
1144 void modify(int x, int y, int z);
1145
1146 void addedge(int x, int y, int z);
1147
1148 void splay(node *x);
1149
1150 void pushup(node *x);
1151
1152 void pushdown(node *x);
1153
1154 void refresh(node *x);
1155
1156 void makeroot(node *x);
1157
1158 void link(node *x, node *y);
1159
1160 void cut(node *x, node *y);
1161
1162 void swap(int &x, int &y);
1163
1164 void deledge(node *x, node *y);
1165
1166 void modify_color(int x);
1167
1168 void modify(int x, int y, int z);
1169
1170 void addedge(int x, int y, int z);
1171
1172 void splay(node *x);
1173
1174 void pushup(node *x);
1175
1176 void pushdown(node *x);
1177
1178 void refresh(node *x);
1179
1180 void makeroot(node *x);
1181
1182 void link(node *x, node *y);
1183
1184 void cut(node *x, node *y);
1185
1186 void swap(int &x, int &y);
1187
1188 void deledge(node *x, node *y);
1189
1190 void modify_color(int x);
1191
1192 void modify(int x, int y, int z);
1193
1194 void addedge(int x, int y, int z);
1195
1196 void splay(node *x);
1197
1198 void pushup(node *x);
1199
1200 void pushdown(node *x);
1201
1202 void refresh(node *x);
1203
1204 void makeroot(node *x);
1205
1206 void link(node *x, node *y);
1207
1208 void cut(node *x, node *y);
1209
1210 void swap(int &x, int &y);
1211
1212 void deledge(node *x, node *y);
1213
1214 void modify_color(int x);
1215
1216 void modify(int x, int y, int z);
1217
1218 void addedge(int x, int y, int z);
1219
1220 void splay(node *x);
1221
1222 void pushup(node *x);
1223
1224 void pushdown(node *x);
1225
1226 void refresh(node *x);
1227
1228 void makeroot(node *x);
1229
1230 void link(node *x, node *y);
1231
1232 void cut(node *x, node *y);
1233
1234 void swap(int &x, int &y);
1235
1236 void deledge(node *x, node *y);
1237
1238 void modify_color(int x);
1239
1240 void modify(int x, int y, int z);
1241
1242 void addedge(int x, int y, int z);
1243
1244 void splay(node *x);
1245
1246 void pushup(node *x);
1247
1248 void pushdown(node *x);
1249
1250 void refresh(node *x);
1251
1252 void makeroot(node *x);
1253
1254 void link(node *x, node *y);
1255
1256 void cut(node *x, node *y);
1257
1258 void swap(int &x, int &y);
1259
1260 void deledge(node *x, node *y);
1261
1262 void modify_color(int x);
1263
1264 void modify(int x, int y, int z);
1265
1266 void addedge(int x, int y, int z);
1267
1268 void splay(node *x);
1269
1270 void pushup(node *x);
1271
1272 void pushdown(node *x);
1273
1274 void refresh(node *x);
1275
1276 void makeroot(node *x);
1277
1278 void link(node *x, node *y);
1279
1280 void cut(node *x, node *y);
1281
1282 void swap(int &x, int &y);
1283
1284 void deledge(node *x, node *y);
1285
1286 void modify_color(int x);
1287
1288 void modify(int x, int y, int z);
1289
1290 void addedge(int x, int y, int z);
1291
1292 void splay(node *x);
1293
1294 void pushup(node *x);
1295
1296 void pushdown(node *x);
1297
1298 void refresh(node *x);
1299
1300 void makeroot(node *x);
1301
1302 void link(node *x, node *y);
1303
1304 void cut(node *x, node *y);
1305
1306 void swap(int &x, int &y);
1307
1308 void deledge(node *x, node *y);
1309
1310 void modify_color(int x);
1311
1312 void modify(int x, int y, int z);
1313
1314 void addedge(int x, int y, int z);
1315
1316 void splay(node *x);
1317
1318 void pushup(node *x);
1319
1320 void pushdown(node *x);
1321
1322 void refresh(node *x);
1323
1324 void makeroot(node *x);
1325
1326 void link(node *x, node *y);
1327
1328 void cut(node *x, node *y);
1329
1330 void swap(int &x, int &y);
1331
1332 void deledge(node *x, node *y);
1333
1334 void modify_color(int x);
1335
1336 void modify(int x, int y, int z);
1337
1338 void addedge(int x, int y, int z);
1339
1340 void splay(node *x);
1341
1342 void pushup(node *x);
1343
1344 void pushdown(node *x);
1345
1346 void refresh(node *x);
1347
1348 void makeroot(node *x);
1349
1350 void link(node *x, node *y);
1351
1352 void cut(node *x, node *y);
1353
1354 void swap(int &x, int &y);
1355
1356 void deledge(node *x, node *y);
1357
1358 void modify_color(int x);
1359
1360 void modify(int x, int y, int z);
1361
1362 void addedge(int x, int y, int z);
1363
1364 void splay(node *x);
1365
1366 void pushup(node *x);
1367
1368 void pushdown(node *x);
1369
1370 void refresh(node *x);
1371
1372 void makeroot(node *x);
1373
1374 void link(node *x, node *y);
1375
1376 void cut(node *x, node *y);
1377
1378 void swap(int &x, int &y);
1379
1380 void deledge(node *x, node *y);
1381
1382 void modify_color(int x);
1383
1384 void modify(int x, int y, int z);
1385
1386 void addedge(int x, int y, int z);
1387
1388 void splay(node *x);
1389
1390 void pushup(node *x);
1391
1392 void pushdown(node *x);
1393
1394 void refresh(node *x);
1395
1396 void makeroot(node *x);
1397
1398 void link
```

```

222 void modify_color(int x) {
223     makeroot(null + x);
224     col[x] ^= true;
225
226     if (col[x])
227         null[x].heap.push(0);
228     else
229         null[x].heap.erase(0);
230
231     heap.erase(null[x].maxsum);
232     null[x].refresh();
233     heap.push(null[x].maxsum);
234 }
235
236 node *newnode(int k) {
237     *(++ptr) = node(k);
238     ptr → ch[0] = ptr → ch[1] = ptr → p = null;
239     return ptr;
240 }
241
242 node *access(node *x) {
243     splay(x);
244     heap.erase(x → maxsum);
245     x → refresh();
246
247     if (x → ch[1] != null) {
248         x → ch[1] → pushdown();
249         x → heap.push(x → ch[1] → prefix);
250         x → refresh();
251         heap.push(x → ch[1] → maxsum);
252     }
253
254     x → ch[1] = null;
255     x → refresh();
256     node *y = x;
257     x = x → p;
258
259     while (x != null) {
260         splay(x);
261         heap.erase(x → maxsum);
262
263         if (x → ch[1] != null) {
264             x → ch[1] → pushdown();
265             x → heap.push(x → ch[1] → prefix);
266             heap.push(x → ch[1] → maxsum);
267         }
268
269         x → heap.erase(y → prefix);
270         x → ch[1] = y;
271         (y = x) → refresh();
272         x = x → p;
273     }
274
275     heap.push(y → maxsum);
276     return y;
277 }
278
279 void makeroot(node *x) {
280     access(x);
281     splay(x);
282     x → rev ^= true;
283 }
284
285 void link(node *x, node *y) { // 新添一条虚边, 维护y对应的堆
286     makeroot(x);
287     makeroot(y);
288
289     x → pushdown();
290     x → p = y;
291     heap.erase(y → maxsum);
292     y → heap.push(x → prefix);

```

```

293     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 void splay(node *x) {
311     x → pushdown();
312
313     while (!isroot(x)) {
314         if (!isroot(x → p))
315             x → p → p → pushdown();
316
317         x → p → pushdown();
318         x → pushdown();
319
320         if (isroot(x → p)) {
321             rot(x → p, dir(x) ^ 1);
322             break;
323         }
324
325         if (dir(x) == dir(x → p))
326             rot(x → p → p, dir(x → p) ^ 1);
327         else
328             rot(x → p, dir(x) ^ 1);
329
330         rot(x → p, dir(x) ^ 1);
331     }
332 }
333
334 void rot(node *x, int d) {
335     node *y = x → ch[d ^ 1];
336
337     if ((x → ch[d ^ 1] = y → ch[d]) != null)
338         y → ch[d] → p = x;
339
340     y → p = x → p;
341
342     if (!isroot(x))
343         x → p → ch[dir(x)] = y;
344
345     (y → ch[d] = x) → p = y;
346
347     x → refresh();
348     y → refresh();
349 }

```

4.8 K-D树

4.8.1 动态K-D树(定期重构)

```

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

```

```

9  bool operator < (const node &a) const {
10    return x[d] < a.x[d];
11  }
12
13  void refresh() {
14    sum = ch[0] → sum + ch[1] → sum + w;
15    l[0] = min(x[0], min(ch[0] → l[0], ch[1] →
16                → l[0]));
17    l[1] = min(x[1], min(ch[0] → l[1], ch[1] →
18                → l[1]));
19    r[0] = max(x[0], max(ch[0] → r[0], ch[1] →
20                → r[0]));
21    r[1] = max(x[1], max(ch[0] → r[1], ch[1] →
22                → r[1]));
23  }
24  null[maxn], *root = null;
25
26  void build(int l, int r, int k, node *&rt) {
27    if (l > r) {
28      rt = null;
29      return;
30    }
31
32    int mid = (l + r) / 2;
33
34    d = k;
35    nth_element(null + l, null + mid, null + r + 1);
36
37    rt = null + mid;
38    build(l, mid - 1, k ^ 1, rt → ch[0]);
39    build(mid + 1, r, k ^ 1, rt → ch[1]);
40
41    rt → refresh();
42
43  void query(node *rt) {
44    if (l[0] ≤ rt → l[0] && l[1] ≤ rt → l[1] && rt
45      → → r[0] ≤ r[0] && rt → r[1] ≤ r[1]) {
46      ans += rt → sum;
47      return;
48    }
49    else if (l[0] > rt → r[0] || l[1] > rt → r[1] ||
50      → r[0] < rt → l[0] || r[1] < rt → l[1])
51      return;
52
53    if (l[0] ≤ rt → x[0] && l[1] ≤ rt → x[1] && rt
54      → → x[0] ≤ r[0] && rt → x[1] ≤ r[1])
55      ans += rt → w;
56
57    query(rt → ch[0]);
58    query(rt → ch[1]);
59
60  int main() {
61
62    null → l[0] = null → l[1] = 100000000;
63    null → r[0] = null → r[1] = -100000000;
64    null → sum = 0;
65    null → ch[0] = null → ch[1] = null;
66    scanf("%*d");
67
68    while (scanf("%d", &op) == 1 && op != 3) {
69      if (op == 1) {
70        tmp++;
71        scanf("%d%d%d", &x[tmp][0], &x[tmp][1],
72              → &w[tmp]);
73        x[tmp][0] ≈ ans;
74        x[tmp][1] ≈ ans;
75        w[tmp] ≈ ans;
76      }
77    }
78  }
79
80  void update(int l, int r, int val) {
81    if (l[0] > r[0] || l[1] > r[1])
82      return;
83    else if (l[0] <= r[0] && l[1] <= r[1])
84      ans += val;
85    else if (l[0] <= r[1] && l[1] <= r[0])
86      ans += val;
87    else if (l[0] <= r[0] && l[1] > r[1])
88      ans += val;
89    else if (l[0] > r[0] && l[1] <= r[1])
90      ans += val;
91    else if (l[0] > r[1] && l[1] > r[0])
92      ans += val;
93    else if (l[0] > r[0] && l[1] > r[1])
94      ans += val;
95  }
96
97  void print() {
98    printf("%d\n", ans);
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
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
745
746
747
747
748
749
749
750
751
752
753
754
755
755
756
757
757
758
759
759
760
761
762
763
764
764
765
766
766
767
767
768
768
769
769
770
771
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
15
```

```

4   bool col[maxn];
5   long long ans_sum;
6   int ans_min, ans_max;
7
8   void add(int x, int y, int z) {
9     G[x].push_back(y);
10    W[x].push_back(z);
11  }
12
13  void dfs(int x) {
14    size[x] = col[x];
15    mx[x] = (col[x] ? d[x] : -inf);
16    mn[x] = (col[x] ? d[x] : inf);
17
18    for (int i = 0; i < (int)G[x].size(); i++) {
19      d[G[x][i]] = d[x] + W[x][i];
20      dfs(G[x][i]);
21      ans_sum += (long long)size[x] * size[G[x]
22        [i]] * d[x];
23      ans_max = max(ans_max, mx[x] + mx[G[x][i]]
24        - (d[x] << 1));
25      ans_min = min(ans_min, mn[x] + mn[G[x][i]]
26        - (d[x] << 1));
27      size[x] += size[G[x][i]];
28      mx[x] = max(mx[x], mx[G[x][i]]);
29      mn[x] = min(mn[x], mn[G[x][i]]);
30    }
31  }
32
33  void clear(int x) {
34    G[x].clear();
35    W[x].clear();
36    col[x] = false;
37  }
38
39  void solve(int rt) {
40    ans_sum = 0;
41    ans_max = -inf;
42    ans_min = inf;
43    dfs(rt);
44    ans_sum <= 1;
45  }
46
47  virtree;
48
49  void dfs(int);
50  int LCA(int, int);
51
52  vector<int> G[maxn];
53  int f[maxn][20], d[maxn], dfn[maxn], tim = 0;
54
55  bool cmp(int x, int y) {
56    return dfn[x] < dfn[y];
57  }
58
59  int n, m, lgn = 0, a[maxn], s[maxn], v[maxn];
60
61  int main() {
62    scanf("%d", &n);
63
64    for (int i = 1, x, y; i < n; i++) {
65      scanf("%d%d", &x, &y);
66      G[x].push_back(y);
67      G[y].push_back(x);
68    }
69
70    G[n + 1].push_back(1);
71    dfs(n + 1);
72
73    for (int i = 1; i <= n + 1; i++)
74      G[i].clear();
75
76    lgn--;
77
78    for (int j = 1; j <= lgn; j++) {
79      for (int i = 1; i <= n; i++)
80        f[i][j] = f[f[i][j - 1]][j - 1];
81
82      scanf("%d", &m);
83
84      while (m--) {
85        int k;
86        scanf("%d", &k);
87
88        for (int i = 1; i <= k; i++)
89          scanf("%d", &a[i]);
90
91        sort(a + 1, a + k + 1, cmp);
92        int top = 0, cnt = 0;
93        s[++top] = v[++cnt] = n + 1;
94        long long ans = 0;
95
96        for (int i = 1; i <= k; i++) {
97          virtree.col[a[i]] = true;
98          ans += d[a[i]] - 1;
99          int u = LCA(a[i], s[top]);
100
101         if (s[top] != u) {
102           while (top > 1 && d[s[top - 1]] >
103             d[u]) {
104             virtree.add(s[top - 1], s[top],
105               d[s[top]] - d[s[top - 1]]);
106             top--;
107           }
108
109           if (s[top] != u) {
110             virtree.add(u, s[top], d[s[top]] -
111               d[u]);
112             s[top] = v[++cnt] = u;
113           }
114
115           s[++top] = a[i];
116         }
117
118         for (int i = top - 1; i; i--)
119           virtree.add(s[i], s[i + 1], d[s[i + 1]] -
120             d[s[i]]);
121
122         virtree.solve(n + 1);
123         ans *= k - 1;
124         printf("%lld %d %d\n", ans - virtree.ans_sum,
125            virtree.ans_min, virtree.ans_max);
126
127         for (int i = 1; i <= k; i++)
128           virtree.clear(a[i]);
129         for (int i = 1; i <= cnt; i++)
130           virtree.clear(v[i]);
131
132       }
133
134       return 0;
135
136       void dfs(int x) {
137         dfn[x] = ++tim;
138         d[x] = d[f[x][0]] + 1;
139
140         while ((1 << lgn) < d[x])
141           lgn++;
142
143         for (int i = 0; i < (int)G[x].size(); i++)
144           if (G[x][i] != f[x][0]) {
145             f[G[x][i]][0] = x;
146             dfs(G[x][i]);
147           }
148       }
149     }
150   }
151 }
```

```
    }

}

int LCA(int x, int y) {
    if (d[x] != d[y]) {
        if (d[x] < d[y])
            swap(x, y);

        for (int i = lgn; i ≥ 0; i--)
            if (((d[x] - d[y]) >> i) & 1)
                x = f[x][i];
    }

    if (x == y)
        return x;

    for (int i = lgn; i ≥ 0; i--)
        if (f[x][i] != f[y][i]) {
            x = f[x][i];
            y = f[y][i];
        }
}

return f[x][0];
}
```

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

4.11 长链剖分

```

1 // 顾名思义，长链剖分是取最深的儿子作为重儿子
2
3 // O(n)维护以深度为下标的子树信息
4 vector<int> G[maxn], v[maxn];
5 int n, p[maxn], h[maxn], son[maxn], ans[maxn];
6
7 // 原题题意：求每个点的子树中与它距离是几的点最多，相同的
8 // 取最大深度
9 // 由于vector只能在后面加入元素，为了写代码方便，这里反过来
10 // 来存
11 // 或者开一个结构体维护倒过来的vector
12 void dfs(int x) {
13     h[x] = 1;
14
15     for (int y : G[x])
16         if (y != p[x]){
17             p[y] = x;
18             dfs(y);
19
20             if (h[y] > h[son[x]])
21                 son[x] = y;
22         }
23
24     if (!son[x]) {
25         v[x].push_back(1);
26         ans[x] = 0;
27         return;
28     }
29
30     h[x] = h[son[x]] + 1;
31     swap(v[x], v[son[x]]);
32
33     if (v[x][ans[son[x]]] == 1)
34         ans[x] = h[x] - 1;
35     else
36         ans[x] = ans[son[x]];
37
38     v[x].push_back(1);
39
40     int mx = v[x][ans[x]];
41     for (int y : G[x])
42         if (y != p[x] && y != son[x]) {
43             for (int i = 1; i <= h[y]; i++)

```

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  $\geq k$ 
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
35 // 第二遍dfs，用于进行剖分和预处理梯子剖分(每条链向上延伸
36 ↪ 一倍)数组
37 void dfs2(int x) {
38     top[x] = (x == son[f[0][x]] ? top[f[0][x]] : x);
39
40     for (int y : G[x])
41         if (y != f[0][x])
42             dfs2(y);
43
44     if (top[x] == x) {
45         int u = x;
46         while (top[son[u]] == x)
47             u = son[u];
48
49         len[x] = d[u] - d[x];
50         for (int i = 0; i < len[x]; i++, u = f[0][u])
51             v[x].push_back(u);
52
53         u = x;
54     }
55 }
56
57 // 求第k祖先
58 int get_k_ancestor(int x, int k) {
59     if (k == 0)
60         return x;
61
62     int l = log_tbl[k];
63     int r = f[l][x];
64
65     if (r == x)
66         return r;
67
68     return get_k_ancestor(r, k - l);
69 }
70
71 // 求第k祖先的深度
72 int get_k_ancestor_depth(int x, int k) {
73     if (k == 0)
74         return d[x];
75
76     int l = log_tbl[k];
77     int r = f[l][x];
78
79     if (r == x)
80         return d[x];
81
82     return get_k_ancestor_depth(r, k - l);
83 }
84
85 // 求第k祖先的深度
86 int get_k_ancestor_depth(int x, int k) {
87     if (k == 0)
88         return d[x];
89
90     int l = log_tbl[k];
91     int r = f[l][x];
92
93     if (r == x)
94         return d[x];
95
96     return get_k_ancestor_depth(r, k - l);
97 }
98
99 // 求第k祖先的深度
100 int get_k_ancestor_depth(int x, int k) {
101     if (k == 0)
102         return d[x];
103
104     int l = log_tbl[k];
105     int r = f[l][x];
106
107     if (r == x)
108         return d[x];
109
110     return get_k_ancestor_depth(r, k - l);
111 }
112
113 // 求第k祖先的深度
114 int get_k_ancestor_depth(int x, int k) {
115     if (k == 0)
116         return d[x];
117
118     int l = log_tbl[k];
119     int r = f[l][x];
120
121     if (r == x)
122         return d[x];
123
124     return get_k_ancestor_depth(r, k - l);
125 }
126
127 // 求第k祖先的深度
128 int get_k_ancestor_depth(int x, int k) {
129     if (k == 0)
130         return d[x];
131
132     int l = log_tbl[k];
133     int r = f[l][x];
134
135     if (r == x)
136         return d[x];
137
138     return get_k_ancestor_depth(r, k - l);
139 }
140
141 // 求第k祖先的深度
142 int get_k_ancestor_depth(int x, int k) {
143     if (k == 0)
144         return d[x];
145
146     int l = log_tbl[k];
147     int r = f[l][x];
148
149     if (r == x)
150         return d[x];
151
152     return get_k_ancestor_depth(r, k - l);
153 }
154
155 // 求第k祖先的深度
156 int get_k_ancestor_depth(int x, int k) {
157     if (k == 0)
158         return d[x];
159
160     int l = log_tbl[k];
161     int r = f[l][x];
162
163     if (r == x)
164         return d[x];
165
166     return get_k_ancestor_depth(r, k - l);
167 }
168
169 // 求第k祖先的深度
170 int get_k_ancestor_depth(int x, int k) {
171     if (k == 0)
172         return d[x];
173
174     int l = log_tbl[k];
175     int r = f[l][x];
176
177     if (r == x)
178         return d[x];
179
180     return get_k_ancestor_depth(r, k - l);
181 }
182
183 // 求第k祖先的深度
184 int get_k_ancestor_depth(int x, int k) {
185     if (k == 0)
186         return d[x];
187
188     int l = log_tbl[k];
189     int r = f[l][x];
190
191     if (r == x)
192         return d[x];
193
194     return get_k_ancestor_depth(r, k - l);
195 }
196
197 // 求第k祖先的深度
198 int get_k_ancestor_depth(int x, int k) {
199     if (k == 0)
200         return d[x];
201
202     int l = log_tbl[k];
203     int r = f[l][x];
204
205     if (r == x)
206         return d[x];
207
208     return get_k_ancestor_depth(r, k - l);
209 }
210
211 // 求第k祖先的深度
212 int get_k_ancestor_depth(int x, int k) {
213     if (k == 0)
214         return d[x];
215
216     int l = log_tbl[k];
217     int r = f[l][x];
218
219     if (r == x)
220         return d[x];
221
222     return get_k_ancestor_depth(r, k - l);
223 }
224
225 // 求第k祖先的深度
226 int get_k_ancestor_depth(int x, int k) {
227     if (k == 0)
228         return d[x];
229
230     int l = log_tbl[k];
231     int r = f[l][x];
232
233     if (r == x)
234         return d[x];
235
236     return get_k_ancestor_depth(r, k - l);
237 }
238
239 // 求第k祖先的深度
240 int get_k_ancestor_depth(int x, int k) {
241     if (k == 0)
242         return d[x];
243
244     int l = log_tbl[k];
245     int r = f[l][x];
246
247     if (r == x)
248         return d[x];
249
250     return get_k_ancestor_depth(r, k - l);
251 }
252
253 // 求第k祖先的深度
254 int get_k_ancestor_depth(int x, int k) {
255     if (k == 0)
256         return d[x];
257
258     int l = log_tbl[k];
259     int r = f[l][x];
260
261     if (r == x)
262         return d[x];
263
264     return get_k_ancestor_depth(r, k - l);
265 }
266
267 // 求第k祖先的深度
268 int get_k_ancestor_depth(int x, int k) {
269     if (k == 0)
270         return d[x];
271
272     int l = log_tbl[k];
273     int r = f[l][x];
274
275     if (r == x)
276         return d[x];
277
278     return get_k_ancestor_depth(r, k - l);
279 }
280
281 // 求第k祖先的深度
282 int get_k_ancestor_depth(int x, int k) {
283     if (k == 0)
284         return d[x];
285
286     int l = log_tbl[k];
287     int r = f[l][x];
288
289     if (r == x)
290         return d[x];
291
292     return get_k_ancestor_depth(r, k - l);
293 }
294
295 // 求第k祖先的深度
296 int get_k_ancestor_depth(int x, int k) {
297     if (k == 0)
298         return d[x];
299
300     int l = log_tbl[k];
301     int r = f[l][x];
302
303     if (r == x)
304         return d[x];
305
306     return get_k_ancestor_depth(r, k - l);
307 }
308
309 // 求第k祖先的深度
310 int get_k_ancestor_depth(int x, int k) {
311     if (k == 0)
312         return d[x];
313
314     int l = log_tbl[k];
315     int r = f[l][x];
316
317     if (r == x)
318         return d[x];
319
320     return get_k_ancestor_depth(r, k - l);
321 }
322
323 // 求第k祖先的深度
324 int get_k_ancestor_depth(int x, int k) {
325     if (k == 0)
326         return d[x];
327
328     int l = log_tbl[k];
329     int r = f[l][x];
330
331     if (r == x)
332         return d[x];
333
334     return get_k_ancestor_depth(r, k - l);
335 }
336
337 // 求第k祖先的深度
338 int get_k_ancestor_depth(int x, int k) {
339     if (k == 0)
340         return d[x];
341
342     int l = log_tbl[k];
343     int r = f[l][x];
344
345     if (r == x)
346         return d[x];
347
348     return get_k_ancestor_depth(r, k - l);
349 }
350
351 // 求第k祖先的深度
352 int get_k_ancestor_depth(int x, int k) {
353     if (k == 0)
354         return d[x];
355
356     int l = log_tbl[k];
357     int r = f[l][x];
358
359     if (r == x)
360         return d[x];
361
362     return get_k_ancestor_depth(r, k - l);
363 }
364
365 // 求第k祖先的深度
366 int get_k_ancestor_depth(int x, int k) {
367     if (k == 0)
368         return d[x];
369
370     int l = log_tbl[k];
371     int r = f[l][x];
372
373     if (r == x)
374         return d[x];
375
376     return get_k_ancestor_depth(r, k - l);
377 }
378
379 // 求第k祖先的深度
380 int get_k_ancestor_depth(int x, int k) {
381     if (k == 0)
382         return d[x];
383
384     int l = log_tbl[k];
385     int r = f[l][x];
386
387     if (r == x)
388         return d[x];
389
390     return get_k_ancestor_depth(r, k - l);
391 }
392
393 // 求第k祖先的深度
394 int get_k_ancestor_depth(int x, int k) {
395     if (k == 0)
396         return d[x];
397
398     int l = log_tbl[k];
399     int r = f[l][x];
400
401     if (r == x)
402         return d[x];
403
404     return get_k_ancestor_depth(r, k - l);
405 }
406
407 // 求第k祖先的深度
408 int get_k_ancestor_depth(int x, int k) {
409     if (k == 0)
410         return d[x];
411
412     int l = log_tbl[k];
413     int r = f[l][x];
414
415     if (r == x)
416         return d[x];
417
418     return get_k_ancestor_depth(r, k - l);
419 }
420
421 // 求第k祖先的深度
422 int get_k_ancestor_depth(int x, int k) {
423     if (k == 0)
424         return d[x];
425
426     int l = log_tbl[k];
427     int r = f[l][x];
428
429     if (r == x)
430         return d[x];
431
432     return get_k_ancestor_depth(r, k - l);
433 }
434
435 // 求第k祖先的深度
436 int get_k_ancestor_depth(int x, int k) {
437     if (k == 0)
438         return d[x];
439
440     int l = log_tbl[k];
441     int r = f[l][x];
442
443     if (r == x)
444         return d[x];
445
446     return get_k_ancestor_depth(r, k - l);
447 }
448
449 // 求第k祖先的深度
450 int get_k_ancestor_depth(int x, int k) {
451     if (k == 0)
452         return d[x];
453
454     int l = log_tbl[k];
455     int r = f[l][x];
456
457     if (r == x)
458         return d[x];
459
460     return get_k_ancestor_depth(r, k - l);
461 }
462
463 // 求第k祖先的深度
464 int get_k_ancestor_depth(int x, int k) {
465     if (k == 0)
466         return d[x];
467
468     int l = log_tbl[k];
469     int r = f[l][x];
470
471     if (r == x)
472         return d[x];
473
474     return get_k_ancestor_depth(r, k - l);
475 }
476
477 // 求第k祖先的深度
478 int get_k_ancestor_depth(int x, int k) {
479     if (k == 0)
480         return d[x];
481
482     int l = log_tbl[k];
483     int r = f[l][x];
484
485     if (r == x)
486         return d[x];
487
488     return get_k_ancestor_depth(r, k - l);
489 }
490
491 // 求第k祖先的深度
492 int get_k_ancestor_depth(int x, int k) {
493     if (k == 0)
494         return d[x];
495
496     int l = log_tbl[k];
497     int r = f[l][x];
498
499     if (r == x)
500         return d[x];
501
502     return get_k_ancestor_depth(r, k - l);
503 }
504
505 // 求第k祖先的深度
506 int get_k_ancestor_depth(int x, int k) {
507     if (k == 0)
508         return d[x];
509
510     int l = log_tbl[k];
511     int r = f[l][x];
512
513     if (r == x)
514         return d[x];
515
516     return get_k_ancestor_depth(r, k - l);
517 }
518
519 // 求第k祖先的深度
520 int get_k_ancestor_depth(int x, int k) {
521     if (k == 0)
522         return d[x];
523
524     int l = log_tbl[k];
525     int r = f[l][x];
526
527     if (r == x)
528         return d[x];
529
530     return get_k_ancestor_depth(r, k - l);
531 }
532
533 // 求第k祖先的深度
534 int get_k_ancestor_depth(int x, int k) {
535     if (k == 0)
536         return d[x];
537
538     int l = log_tbl[k];
539     int r = f[l][x];
540
541     if (r == x)
542         return d[x];
543
544     return get_k_ancestor_depth(r, k - l);
545 }
546
547 // 求第k祖先的深度
548 int get_k_ancestor_depth(int x, int k) {
549     if (k == 0)
550         return d[x];
551
552     int l = log_tbl[k];
553     int r = f[l][x];
554
555     if (r == x)
556         return d[x];
557
558     return get_k_ancestor_depth(r, k - l);
559 }
560
561 // 求第k祖先的深度
562 int get_k_ancestor_depth(int x, int k) {
563     if (k == 0)
564         return d[x];
565
566     int l = log_tbl[k];
567     int r = f[l][x];
568
569     if (r == x)
570         return d[x];
571
572     return get_k_ancestor_depth(r, k - l);
573 }
574
575 // 求第k祖先的深度
576 int get_k_ancestor_depth(int x, int k) {
577     if (k == 0)
578         return d[x];
579
580     int l = log_tbl[k];
581     int r = f[l][x];
582
583     if (r == x)
584         return d[x];
585
586     return get_k_ancestor_depth(r, k - l);
587 }
588
589 // 求第k祖先的深度
590 int get_k_ancestor_depth(int x, int k) {
591     if (k == 0)
592         return d[x];
593
594     int l = log_tbl[k];
595     int r = f[l][x];
596
597     if (r == x)
598         return d[x];
599
600     return get_k_ancestor_depth(r, k - l);
601 }
602
603 // 求第k祖先的深度
604 int get_k_ancestor_depth(int x, int k) {
605     if (k == 0)
606         return d[x];
607
608     int l = log_tbl[k];
609     int r = f[l][x];
610
611     if (r == x)
612         return d[x];
613
614     return get_k_ancestor_depth(r, k - l);
615 }
616
617 // 求第k祖先的深度
618 int get_k_ancestor_depth(int x, int k) {
619     if (k == 0)
620         return d[x];
621
622     int l = log_tbl[k];
623     int r = f[l][x];
624
625     if (r == x)
626         return d[x];
627
628     return get_k_ancestor_depth(r, k - l);
629 }
630
631 // 求第k祖先的深度
632 int get_k_ancestor_depth(int x, int k) {
633     if (k == 0)
634         return d[x];
635
636     int l = log_tbl[k];
637     int r = f[l][x];
638
639     if (r == x)
640         return d[x];
641
642     return get_k_ancestor_depth(r, k - l);
643 }
644
645 // 求第k祖先的深度
646 int get_k_ancestor_depth(int x, int k) {
647     if (k == 0)
648         return d[x];
649
650     int l = log_tbl[k];
651     int r = f[l][x];
652
653     if (r == x)
654         return d[x];
655
656     return get_k_ancestor_depth(r, k - l);
657 }
658
659 // 求第k祖先的深度
660 int get_k_ancestor_depth(int x, int k) {
661     if (k == 0)
662         return d[x];
663
664     int l = log_tbl[k];
665     int r = f[l][x];
666
667     if (r == x)
668         return d[x];
669
670     return get_k_ancestor_depth(r, k - l);
671 }
672
673 // 求第k祖先的深度
674 int get_k_ancestor_depth(int x, int k) {
675     if (k == 0)
676         return d[x];
677
678     int l = log_tbl[k];
679     int r = f[l][x];
680
681     if (r == x)
682         return d[x];
683
684     return get_k_ancestor_depth(r, k - l);
685 }
686
687 // 求第k祖先的深度
688 int get_k_ancestor_depth(int x, int k) {
689     if (k == 0)
690         return d[x];
691
692     int l = log_tbl[k];
693     int r = f[l][x];
694
695     if (r == x)
696         return d[x];
697
698     return get_k_ancestor_depth(r, k - l);
699 }
700
701 // 求第k祖先的深度
702 int get_k_ancestor_depth(int x, int k) {
703     if (k == 0)
704         return d[x];
705
706     int l = log_tbl[k];
707     int r = f[l][x];
708
709     if (r == x)
710         return d[x];
711
712     return get_k_ancestor_depth(r, k - l);
713 }
714
715 // 求第k祖先的深度
716 int get_k_ancestor_depth(int x, int k) {
717     if (k == 0)
718         return d[x];
719
720     int l = log_tbl[k];
721     int r = f[l][x];
722
723     if (r == x)
724         return d[x];
725
726     return get_k_ancestor_depth(r, k - l);
727 }
728
729 // 求第k祖先的深度
730 int get_k_ancestor_depth(int x, int k) {
731     if (k == 0)
732         return d[x];
733
734     int l = log_tbl[k];
735     int r = f[l][x];
736
737     if (r == x)
738         return d[x];
739
740     return get_k_ancestor_depth(r, k - l);
741 }
742
743 // 求第k祖先的深度
744 int get_k_ancestor_depth(int x, int k) {
745     if (k == 0)
746         return d[x];
747
748     int l = log_tbl[k];
749     int r = f[l][x];
750
751     if (r == x)
752         return d[x];
753
754     return get_k_ancestor_depth(r, k - l);
755 }
756
757 // 求第k祖先的深度
758 int get_k_ancestor_depth(int x, int k) {
759     if (k == 0)
760         return d[x];
761
762     int l = log_tbl[k];
763     int r = f[l][x];
764
765     if (r == x)
766         return d[x];
767
768     return get_k_ancestor_depth(r, k - l);
769 }
770
771 // 求第k祖先的深度
772 int get_k_ancestor_depth(int x, int k) {
773     if (k == 0)
774         return d[x];
775
776     int l = log_tbl[k];
777     int r = f[l][x];
778
779     if (r == x)
780         return d[x];
781
782     return get_k_ancestor_depth(r, k - l);
783 }
784
785 // 求第k祖先的深度
786 int get_k_ancestor_depth(int x, int k) {
787     if (k == 0)
788         return d[x];
789
790     int l = log_tbl[k];
791     int r = f[l][x];
792
793     if (r == x)
794         return d[x];
795
796     return get_k_ancestor_depth(r, k - l);
797 }
798
799 // 求第k祖先的深度
800 int get_k_ancestor_depth(int x, int k) {
801     if (k == 0)
802         return d[x];
803
804     int l = log_tbl[k];
805     int r = f[l][x];
806
807     if (r == x)
808         return d[x];
809
810     return get_k_ancestor_depth(r, k - l);
811 }
812
813 // 求第k祖先的深度
814 int get_k_ancestor_depth(int x, int k) {
815     if (k == 0)
816         return d[x];
817
818     int l = log_tbl[k];
819     int r = f[l][x];
820
821     if (r == x)
822         return d[x];
823
824     return get_k_ancestor_depth(r, k - l);
825 }
826
827 // 求第k祖先的深度
828 int get_k_ancestor_depth(int x, int k) {
829     if (k == 0)
830         return d[x];
831
832     int l = log_tbl[k];
833     int r = f[l][x];
834
835     if (r == x)
836         return d[x];
837
838     return get_k_ancestor_depth(r, k - l);
839 }
840
841 // 求第k祖先的深度
842 int get_k_ancestor_depth(int x, int k) {
843     if (k == 0)
844         return d[x];
845
846     int l = log_tbl[k];
847     int r = f[l][x];
848
849     if (r == x)
850         return d[x];
851
852     return get_k_ancestor_depth(r, k - l);
853 }
854
855 // 求第k祖先的深度
856 int get_k_ancestor_depth(int x, int k) {
857     if (k == 0)
858         return d[x];
859
860     int l = log_tbl[k];
861     int r = f[l][x];
862
863     if (r == x)
864         return d[x];
865
866     return get_k_ancestor_depth(r, k - l);
867 }
868
869 // 求第k祖先的深度
870 int get_k_ancestor_depth(int x, int k) {
871     if (k == 0)
872         return d[x];
873
874     int l = log_tbl[k];
875     int r = f[l][x];
876
877     if (r == x)
878         return d[x];
879
880     return get_k_ancestor_depth(r, k - l);
881 }
882
883 // 求第k祖先的深度
884 int get_k_ancestor_depth(int x, int k) {
885     if (k == 0)
886         return d[x];
887
888     int l = log_tbl[k];
889     int r = f[l][x];
890
891     if (r == x)
892         return d[x];
893
894     return get_k_ancestor_depth(r, k - l);
895 }
896
897 // 求第k祖先的深度
898 int get_k_ancestor_depth(int x, int k) {
899     if (k == 0)
900         return d[x];
901
902     int l = log_tbl[k];
903     int r = f[l][x];
904
905     if (r == x)
906         return d[x];
907
908     return get_k_ancestor_depth(r, k - l);
909 }
910
911 // 求第k祖先的深度
912 int get_k_ancestor_depth(int x, int k) {
913     if (k == 0)
914         return d[x];
915
916     int l = log_tbl[k];
917     int r = f[l][x];
918
919     if (r == x)
920         return d[x];
921
922     return get_k_ancestor_depth(r, k - l);
923 }
924
925 // 求第k祖先的深度
926 int get_k_ancestor_depth(int x, int k) {
927     if (k == 0)
928         return d[x];
929
930     int l = log_tbl[k];
931     int r = f[l][x];
932
933     if (r == x)
934         return d[x];
935
936     return get_k_ancestor_depth(r, k - l);
937 }
938
939 // 求第k祖先的深度
940 int get_k_ancestor_depth(int x, int k) {
941     if (k == 0)
942         return d[x];
943
944     int l = log_tbl[k];
945     int r = f[l][x];
946
947     if (r == x)
948         return d[x];
949
950     return get_k_ancestor_depth(r, k - l);
951 }
952
953 // 求第k祖先的深度
954 int get_k_ancestor_depth(int x, int k) {
955     if (k == 0)
956         return d[x];
957
958     int l = log_tbl[k];
959     int r = f[l][x];
960
961     if (r == x)
962         return d[x];
963
964     return get_k_ancestor_depth(r, k - l);
965 }
966
967 // 求第k祖先的深度
968 int get_k_ancestor_depth(int x, int k) {
969     if (k == 0)
970         return d[x];
971
972     int l = log_tbl[k];
973     int r = f[l][x];
974
975     if (r == x)
976         return d[x];
977
978     return get_k_ancestor_depth(r, k - l);
979 }
980
981 // 求第k祖先的深度
982 int get_k_ancestor_depth(int x, int k) {
983     if (k == 0)
984         return d[x];
985
986     int l = log_tbl[k];
987     int r = f[l][x];
988
989     if (r == x)
990         return d[x];
991
992     return get_k_ancestor_depth(r, k - l);
993 }
994
995 // 求第k祖先的深度
996 int get_k_ancestor_depth(int x, int k) {
997     if (k == 0)
998         return d[x];
999
1000    int l = log_tbl[k];
1001    int r = f[l][x];
1002
1003    if (r == x)
1004        return d[x];
1005
1006    return get_k_ancestor_depth(r, k - l);
1007 }
1008
1009 // 求第k祖先的深度
1010 int get_k_ancestor_depth(int x, int k) {
1011     if (k == 0)
1012         return d[x];
1013
1014     int l = log_tbl[k];
1015     int r = f[l][x];
1016
1017     if (r == x)
1018         return d[x];
1019
1020     return get_k_ancestor_depth(r, k - l);
1021 }
1022
1023 // 求第k祖先的深度
1024 int get_k_ancestor_depth(int x, int k) {
1025     if (k == 0)
1026         return d[x];
1027
1028     int l = log_tbl[k];
1029     int r = f[l][x];
1030
1031     if (r == x)
1032         return d[x];
1033
1034     return get_k_ancestor_depth(r, k - l);
1035 }
1036
1037 // 求第k祖先的深度
1038 int get_k_ancestor_depth(int x, int k) {
1039     if (k == 0)
1040         return d[x];
1041
1042     int l = log_tbl[k];
1043     int r = f[l][x];
1044
1045     if (r == x)
1046         return d[x];
1047
1048     return get_k_ancestor_depth(r, k - l);
1049 }
1050
1051 // 求第k祖先的深度
1052 int get_k_ancestor_depth(int x, int k) {
1053     if (k == 0)
1054         return d[x];
1055
1056     int l = log_tbl[k];
1057     int r = f[l][x];
1058
1059     if (r == x)
1060         return d[x];
1061
1062     return get_k_ancestor_depth(r, k - l);
1063 }
1064
1065 // 求第k祖先的深度
1066 int get_k_ancestor_depth(int x, int k) {
1067     if (k == 0)
1068         return d[x];
1069
1070     int l = log_tbl[k];
1071     int r = f[l][x];
1072
1073     if (r == x)
1074         return d[x];
1075
1076     return get_k_ancestor_depth(r, k - l);
1077 }
1078
1079 // 求第k祖先的深度
1080 int get_k_ancestor_depth(int x, int k) {
1081     if (k == 0)
1082         return d[x];
1083
1084     int l = log_tbl[k];
1085     int r = f[l][x];
1086
1087     if (r == x)
1088         return d[x];
1089
1090     return get_k_ancestor_depth(r, k - l);
1091 }
1092
1093 // 求第k祖先的深度
1094 int get_k_ancestor_depth(int x, int k) {
1095     if (k == 0)
1096         return d[x];
1097
1098     int l = log_tbl[k];
1099     int r = f[l][x];
1100
1101     if (r == x)
1102         return d[x];
1103
1104     return get_k_ancestor_depth(r, k - l);
1105 }
1106
1107 // 求第k祖先的深度
1108 int get_k_ancestor_depth(int x, int k) {
1109     if (k == 0)
1110         return d[x];
1111
1112     int l = log_tbl[k];
1113     int r = f[l][x];
1114
1115     if (r == x)
1116         return d[x];
1117
1118     return get_k_ancestor_depth(r, k - l);
1119 }
1120
1121 // 求第k祖先的深度
1122 int get_k_ancestor_depth(int x, int k) {
1123     if (k == 0)
1124         return d[x];
1125
1126     int l = log_tbl[k];
1127     int r = f[l][x];
1128
1129     if (r == x)
1130         return d[x];
1131
1132     return get_k_ancestor_depth(r, k - l);
1133 }
1134
1135 // 求第k祖先的深度
1136 int get_k_ancestor_depth(int x, int k) {
1137     if (k == 0)
1138         return d[x];
1139
1140     int l = log_tbl[k];
1141     int r = f[l][x];
1142
1143     if (r == x)
1144         return d[x];
1145
1146     return get_k_ancestor_depth(r, k - l);
1147 }
1148
1149 // 求第k祖先的深度
1150 int get_k_ancestor_depth(int x, int k) {
1151     if (k == 0)
1152         return d[x];
1153
1154     int l = log_tbl[k];
1155     int r = f[l][x];
1156
1157     if (r == x)
1158         return d[x];
1159
1160     return get_k_ancestor_depth(r, k - l);
1161 }
1162
1163 // 求第k祖先的深度
1164 int get_k_ancestor_depth(int x, int k) {
1165     if (k == 0)
1166         return d[x];
1167
1168     int l = log_tbl[k];
1169     int r = f[l][x];
1170
1171     if (r == x)
1172         return d[x];
1173
1174     return get_k_ancestor_depth(r, k - l);
1175 }
1176
1177 // 求第k祖先的深度
1178 int get_k_ancestor_depth(int x, int k) {
1179     if (k == 0)
1180         return d[x];
1181
1182     int l = log_tbl[k];
1183     int r = f[l][x];
1184
1185     if (r == x)
1186         return d[x];
1187
1188     return get_k_ancestor_depth(r, k - l);
1189 }
1190
1191 // 求第k祖先的深度
1192 int get_k_ancestor_depth(int x, int k) {
1193     if (k == 0)
1194         return d[x];
1195
1196     int l = log_tbl[k];
1197     int r = f[l][x];
1198
1199     if (r == x)
1200         return d[x];
1201
1202     return get_k_ancestor_depth(r, k - l);
1203 }
1204
1205 // 求第k祖先的深度
1206 int get_k_ancestor_depth(int x, int k) {
1207     if (k == 0)
1208         return d[x];
1209
1210     int l = log_tbl[k];
1211     int r = f[l][x];
1212
1213     if (r == x)
1214         return d[x];
1215
1216     return get_k_ancestor_depth(r, k - l);
1217 }
1218
1219 // 求第k祖先的深度
1220 int get_k_ancestor_depth(int x, int k) {
1221     if (k == 0)
1222         return d[x];
1223
1224     int l = log_tbl[k];
1225     int r = f[l][x];
1226
1227     if (r == x)
1228         return d[x];
1229
1230     return get_k_ancestor_depth(r, k - l);
1231 }
1232
1233 // 求第k祖先的深度
1234 int get_k_ancestor_depth(int x, int k) {
1235     if (k == 0)
1236         return d[x];
1237
1238     int l = log_tbl[k];
1239     int r = f[l][x];
1240
1241     if (r == x)
1242         return d[x];
1243
1244     return get_k_ancestor_depth(r, k - l);
1245 }
1246
1247 // 求第k祖先的深度
1248 int get_k_ancestor_depth(int x, int k) {
1249     if (k == 0)
1250         return d[x];
1251
1252     int l = log_tbl[k];
1253     int r = f[l][x];
1254
1255     if (r == x)
1256         return d[x];
1257
1258     return get_k_ancestor_depth(r, k - l);
1259 }
1260
1261 // 求第k祖先的深度
1262 int get_k_ancestor_depth(int x, int k) {
1263     if (k == 0)
1264         return d[x];
1265
1266     int l = log_tbl[k];
1267     int r = f[l][x];
1268
1269     if (r == x)
1270         return d[x];
1271
1272     return get_k_ancestor_depth(r, k - l);
1273 }
1274
1275 // 求第k祖先的深度
1276 int get_k_ancestor_depth(int x, int k) {
1277     if (k == 0)
1278         return d[x];
1279
1280     int l = log_tbl[k];
1281     int r = f[l][x];
1282
1283     if (r == x)
1284         return d[x];
1285
1286     return get_k_ancestor_depth(r, k - l);
1287 }
1288
1289 // 求第k祖先的深度
1290 int get_k_ancestor_depth(int x, int k) {
1291     if (k == 0)
1292         return d[x];
1293
1294     int l = log_tbl[k];
1295     int r = f[l][x];
1296
1297     if (r == x)
1298         return d[x];
1299
1300     return get_k_ancestor_depth(r, k - l);
1301 }
1302
1303 // 求第k祖先的深度
1304 int get_k_ancestor_depth(int x, int k) {
1305     if (k == 0)
1306         return d[x];
1307
1308     int l = log_tbl[k];
1309     int r = f[l][x];
1310
1311     if (r == x)
1312         return d[x];
1313
1314     return get_k_ancestor_depth(r, k - l);
1315 }
1316
1317 // 求第k祖先的深度
1318 int get_k_ancestor_depth(int x, int k) {
1319     if (k == 0)
1320         return d[x];
1321
1322     int l = log_tbl[k];
1323     int r = f[l][x];
1324
1325     if (r == x)
1326         return d[x];
1327
1328     return get_k_ancestor_depth(r, k - l);
1329 }
1330
1331 // 求第k祖先的深度
1332 int get_k_ancestor_depth(int x, int k) {
1333     if (k == 0)
1334         return d[x];
1335
1336     int l = log_tbl[k];
1337
```

```

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

```

4.12 堆

4.12.1 左偏树

参见3.2.3.k短路(27页).

4.12.2 二叉堆

```

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

```

```

42             ↪
43         return res;
44     }
45 }

```

4.13 莫队

注意如果 n 和 q 不平衡, 块大小应该设为 $\frac{n}{\sqrt{q}}$.

另外如果裸的莫队要卡常可以按块编号奇偶性分别对右端点正序或者倒序排序, 期望可以减少一半的移动次数.

4.13.1 莫队二次离线

适用范围: 询问的是点对相关(或者其它可以枚举每个点和区间算贡献)的信息, 并且可以离线; 更新时可以使用一些牺牲修改复杂度来改善询问复杂度的数据结构(如单点修改询问区间和).

先按照普通的莫队将区间排序. 考虑区间移动的情况, 以 (l, r) 向右移动右端点到 (l, t) 为例.

对于每个 $i \in (r, t]$ 来说, 它都要对区间 $[l, i]$ 算贡献. 可以拆成 $[1, i]$ 和 $[1, l]$ 两部分, 那么前一部分因为都是 i 和 $[1, i]$ 做贡献的形式所以可以直接预处理.

考虑后一部分, i 和 $[1, l]$ 做贡献, 因为莫队的性质我们可以保证这样的询问次数不超过 $O((n + m)\sqrt{n})$, 因此我们可以对每个 l 记录下来哪些 i 要和它询问. 并且每次移动时询问的 i 都是连续的, 所以对每个 l 开一个vector记录下对应的区间和编号就行了.

剩余的三种情况(右端点左移或者移动左端点)都是类似的, 具体可以看代码.

例: Yuno loves sqrt technology II (询问区间逆序对数)

```

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

```

```

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

```

```

107
108     l = s;
109
110     if (t > r)
111         vp[l - 1].push_back(Q(r + 1, t, 1, i));
112     else if (t < r)
113         vp[l - 1].push_back(Q(t + 1, r, -1, i));
114
115     r = t;
116 }
117
118 for (int i = 1; i <= n; i++) { // 第一遍正着处理, 解
119     → 决关于前缀的询问
120     fp[i] = fp[i - 1] + querys(a[i] + 1);
121
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     }

```

4.13.2 带修莫队在线化 $O(n^{\frac{5}{3}})$

最简单的带修莫队：块大小设成 $n^{\frac{2}{3}}$ ，排序时第一关键字是左端点块编号，第二关键字是右端点块编号，第三关键字是时间。（记得把时间压缩成只有修改的时间。）

现在要求在线的同时支持修改，仍然以 $B = n^{\frac{2}{3}}$ 分一块，预处理出两块之间的贡献，那么预处理复杂度就是 $O(n^{\frac{5}{3}})$ 。

修改时最简单的方法是直接把 $n^{\frac{2}{3}}$ 个区间全更新一遍。嫌慢的话可以给每个区间打一个懒标记，询问的时候如果解了再更新区间的信息。

注意如果附加信息是可减的（比如每个数的出现次数），那么就只需要存 $O(n^{\frac{1}{3}})$ 个。

总复杂度仍然是 $O(n^{\frac{5}{3}})$ ，如果打懒标记的话是跑不太满的。如果附加信息可减则空间复杂度是 $O(n^{\frac{4}{3}})$ ，否则和时间复杂度同阶。

4.13.3 莫队二次离线 在线化 $O((n+m)\sqrt{n})$

和之前的道理是一样的， i 和 $[1, i]$ 的贡献这部分仍然可以预处理掉，而前后缀对区间的贡献那部分只保存块端点处的信息。

按照莫队二次离线的转移方法操作之后发现只剩两边散块的贡献没有解决。这时可以具体问题具体解决，例如求逆序对的话直接预处理出排序后的数组然后归并即可。

时空复杂度均为 $O(n\sqrt{n})$ 。

以下代码以强制在线求区间逆序对为例（洛谷上被卡常了，正常情况下极限数据应该在 1.5s 内。）

```

1 constexpr int maxn = 100005, B = 315, maxb = maxn / B +
2   ↪ 5;
3
4 int n, bid[maxn], L[maxb], R[maxb], cntb;
5
6 struct DS { // O(sqrt(n)) 修改 O(1) 查询
7     int total;
8     int sa[maxn], sb[maxb];
9
10    void init(const DS &o) {
11        total = o.total;
12        memcpy(sa, o.sa, sizeof(int) * (n + 1));
13        memcpy(sb, o.sb, sizeof(int) * (cntb + 1));
14    }
15
16    void add(int x) {
17        total++;
18        for (int k = 1; k ≤ bid[x]; k++)
19            sb[k]++;
20        for (int i = L[bid[x]]; i ≤ x; i++)
21            sa[i]++;
22    }
23
24    int querys(int x) {
25        if (x > n)
26            return 0;
27
28        return sb[bid[x] + 1] + sa[x];
29    }
30
31    int queryp(int x) {
32        return total - querys(x + 1);
33    }
34 } pr[maxb];
35
36 int c[maxn]; // 树状数组
37
38 void addc(int x, int d) {
39     while (x) {
40         c[x] += d;
41         x -= x & -x;
42     }
43 }
```

```

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 rk[maxn], val[maxn][B + 5];
56
57 long long dat[maxb][maxb];
58
59 int a[maxn];
60
61 int main() {
62     int m;
63     cin >> n >> m;
64
65     for (int i = 1; i ≤ n; i++) {
66         cin >> a[i];
67
68         bid[i] = (i - 1) / B + 1;
69         if (!L[bid[i]])
70             L[bid[i]] = i;
71         R[bid[i]] = i;
72         cntb = bid[i];
73
74         rk[i] = i;
75     }
76
77
78     for (int k = 1; k ≤ cntb; k++)
79         sort(rk + L[k], rk + R[k] + 1, [](int x, int
80           ↪ y) {return a[x] < a[y];}); // 每个块排序
81
82     for (int i = n; i; i--) {
83         for (int j = 2; i + j - 1 ≤ R[bid[i]]; j++) {
84             val[i][j] = val[i + 1][j - 1] + val[i][j -
85               ↪ 1] - val[i + 1][j - 2];
86             if (a[i] > a[i + j - 1])
87                 val[i][j]++; // 块内用二维前缀和预处理
88         }
89
90         for (int k = 1; k ≤ cntb; k++) {
91             for (int i = L[k]; i ≤ R[k]; i++) {
92                 dat[k][k] += queryc(a[i] + 1); // 单块内的逆
93                 addc(a[i], 1);
94             }
95             for (int i = L[k]; i ≤ R[k]; i++)
96                 addc(a[i], -1);
97         }
98
99         for (int i = 1; i ≤ n; i++) {
100             if (i > 1 & i == L[bid[i]])
101                 pr[bid[i]].init(pr[bid[i] - 1]);
102             fp[i] = fp[i - 1] + pr[bid[i]].querys(a[i] +
103               ↪ 1);
104             pr[bid[i]].addc(a[i]);
105         }
106
107         for (int i = n; i; i--) {
108             fs[i] = fs[i + 1] + (n - i - queryc(a[i] + 1));
109         }
110     }
111 }
```

```

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

4.14 常见根号思路

1. 通用

- 出现次数大于 \sqrt{n} 的数不会超过 \sqrt{n} 个
- 对于带修改问题, 如果不方便分治或者二进制分组, 可以考虑对操作分块, 每次查询时暴力最后的 \sqrt{n} 个修改并更正答案
- **根号分治:** 如果分治时每个子问题需要 $O(N)$ (N 是全局问题的大小)的时间, 而规模较小的子问题可以 $O(n^2)$ 解决, 则可以使用根号分治
 - 规模大于 \sqrt{n} 的子问题用 $O(N)$ 的方法解决, 规模小于 \sqrt{n} 的子问题用 $O(n^2)$ 暴力
 - 规模大于 \sqrt{n} 的子问题最多只有 \sqrt{n} 个
 - 规模不大于 \sqrt{n} 的子问题大小的平方和也必定不会超过 $n\sqrt{n}$
- 如果输入规模之和不大于 n (例如给定多个小字符串与大字符串进行询问), 那么规模超过 \sqrt{n} 的问题最多只有 \sqrt{n} 个

2. 序列

- 某些维护序列的问题可以用分块/块状链表维护
- 对于静态区间询问问题, 如果可以快速将左/右端点移动一位, 可以考虑莫队
 - 如果强制在线可以分块预处理, 但是一般空间需要 $n\sqrt{n}$
 - * 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 char s[maxn], t[maxn];
2 int fail[maxn];
3 int n, m;
4
5 void init() { // 注意字符串是0-based, 但是fail是1-based
6     // memset(fail, 0, sizeof(fail));
7
8     for (int i = 1; i < m; i++) {
9         int j = fail[i];
10        while (j && t[i] != t[j])
11            j = fail[j];
12
13        if (t[i] == t[j])
14            fail[i + 1] = j + 1;
15        else
16            fail[i + 1] = 0;
17    }
18}
19
20 int KMP() {
21     int cnt = 0, j = 0;
22
23     for (int i = 0; i < n; i++) {
24         while (j && s[i] != t[j])
25             j = fail[j];
26
27         if (s[i] == t[j])
28             j++;
29         if (j == m)
30             cnt++;
31     }
32
33     return cnt;
34}

```

5.1.1 ex-KMP

```

1 //全局变量与数组定义
2 char s[maxn], t[maxn];
3 int n, m, a[maxn];
4
5 // 主过程 O(n + m)
6 // 把t的每个后缀与s的LCP输出到a中, s的后缀和自己的LCP存
7 // 在nx中
8 // 0-based, s的长度是m, t的长度是n
9 void exKMP(const char *s, const char *t, int *a) {
10     static int nx[maxn];
11
12     memset(nx, 0, sizeof(nx));
13
14     int j = 0;
15     while (j + 1 < m && s[j] == s[j + 1])
16         j++;
17     nx[1] = j;
18
19     for (int i = 2, k = 1; i < m; i++) {
20         int pos = k + nx[k], len = nx[i - k];
21
22         if (i + len < pos)
23             nx[i] = len;
24         else {
25             j = max(pos - i, 0);
26             while (i + j < m && s[j] == s[i + j])
27                 j++;
28             nx[i] = j;
29         }
30     }
31 }

```

```

27
28         nx[i] = j;
29         k = i;
30     }
31 }
32
33 j = 0;
34 while (j < n && j < m && s[j] == t[j])
35     j++;
36 a[0] = j;
37
38 for (int i = 1, k = 0; i < n; i++) {
39     int pos = k + a[k], len = nx[i - k];
40     if (i + len < pos)
41         a[i] = len;
42     else {
43         j = max(pos - i, 0);
44         while (j < m && i + j < n && s[j] == t[i + j])
45             j++;
46         a[i] = j;
47         k = i;
48     }
49 }
50
51

```

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
27         G[f[x][0]].push_back(x);
28         fill(f[x] + 1, f[x] + 26, cnt + 1);
29
30         for (int c = 0; c < 26; c++) {
31             if (ch[x][c]) {
32                 int y = f[x][0];
33
34                 f[ch[x][c]][0] = ch[y][c];
35                 q[tail++] = ch[x][c];
36             }
37             else
38                 ch[x][c] = ch[f[x][0]][c];
39         }
40     }
41     fill(f[0], f[0] + 26, cnt + 1);
42 }

```

40

}

5.3 后缀数组

5.3.1 倍增

```

1 constexpr int maxn = 100005;
2
3 void get_sa(char *s, int n, int *sa, int *rnk, int
4   ↪ *height) { // 1-base
5   static int buc[maxn], id[maxn], p[maxn], t[maxn *
6     ↪ 2];
7
8   int m = 300;
9
10  for (int i = 1; i ≤ n; i++)
11    | buc[rnk[i]] = s[i]++;
12  for (int i = 1; i ≤ m; i++)
13    | buc[i] += buc[i - 1];
14  for (int i = n; i; i--)
15    | sa[buc[rnk[i]]--] = i;
16
17  memset(buc, 0, sizeof(int) * (m + 1));
18
19  for (int k = 1, cnt = 0; cnt != n; k *= 2, m = cnt)
20    ↪ {
21    | cnt = 0;
22    | for (int i = n; i > n - k; i--)
23      |   id[++cnt] = i;
24
25    | for (int i = 1; i ≤ n; i++)
26      |   if (sa[i] > k)
27        |     id[++cnt] = sa[i] - k;
28
29    | for (int i = 1; i ≤ n; i++)
30      |   buc[p[i]] = rnk[id[i]]++;
31    for (int i = 1; i ≤ m; i++)
32      |   buc[i] += buc[i - 1];
33    for (int i = n; i; i--)
34      |   sa[buc[p[i]]--] = id[i];
35
36    memset(buc, 0, sizeof(int) * (m + 1));
37
38    memcpy(t, rnk, sizeof(int) * (max(n, m) + 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      | sa[rnk[i]] = i;
51
52    for (int i = 1, k = 0; i ≤ n; i++) { // 顺便
53      ↪ 求height
54      | if (k)
55        |   k--;
56
57      while (s[i + k] == s[sa[rnk[i] - 1] + k])
58        |   k++;
59
60      height[rnk[i]] = k; // height[i] = lcp(sa[i],
61        ↪ sa[i - 1])
62
63    }
64
65  }
```

58 }

```

59
60 char s[maxn];
61 int sa[maxn], rnk[maxn], height[maxn];
62
63 int main() {
64   cin >> (s + 1);
65
66   int n = strlen(s + 1);
67
68   get_sa(s, n, sa, rnk, height);
69
70   for (int i = 1; i ≤ n; i++)
71     cout << sa[i] << (i < n ? ' ' : '\n');
72
73   for (int i = 2; i ≤ n; i++)
74     cout << height[i] << (i < n ? ' ' : '\n');
75
76   return 0;
77 }
```

5.3.2 SA-IS

```

1 // SA-IS求完的SA有效位只有1~n, 但它是0-based, 如果其他部
2   ↪ 分是1-based就抄一下封装
3
4 constexpr int maxn = 100005, l_type = 0, s_type = 1;
5
6 // 判断一个字符是否为LMS字符
7 bool is_lms(int *tp, int x) {
8   return x > 0 && tp[x] == s_type && tp[x - 1] ==
9     ↪ l_type;
10
11 // 判断两个LMS子串是否相同
12 bool equal_substr(int *s, int x, int y, int *tp) {
13   do {
14     if (s[x] != s[y])
15       return false;
16     x++;
17     y++;
18   } while (!is_lms(tp, x) && !is_lms(tp, y));
19
20   return s[x] == s[y];
21
22 // 诱导排序(从*型诱导到L型, 从L型诱导到S型)
23 // 调用之前应将*型按要求放入SA中
24 void induced_sort(int *s, int *sa, int *tp, int *buc,
25   ↪ int *lbuc, int *sbuc, int n, int m) {
26   for (int i = 0; i ≤ n; i++)
27     if (sa[i] > 0 && tp[sa[i] - 1] == l_type)
28       sa[lbuc[s[sa[i] - 1]]++] = sa[i] - 1;
29
30   for (int i = 1; i ≤ m; i++)
31     sbuc[i] = buc[i] - 1;
32
33   for (int i = n; ~i; i--)
34     if (sa[i] > 0 && tp[sa[i] - 1] == s_type)
35       sa[sbuc[s[sa[i] - 1]]--] = sa[i] - 1;
36
37 // s是输入字符串, n是字符串的长度, m是字符集的大小
38 int *sa_is(int *s, int len, int m) {
39   int n = len - 1;
40
41   int *tp = new int[n + 1];
42   int *pos = new int[n + 1];
43   int *name = new int[n + 1];
44 }
```

```

44 int *sa = new int[n + 1];
45 int *buc = new int[m + 1];
46 int *lbuc = new int[m + 1];
47 int *sbuc = new int[m + 1];
48
49 memset(buc, 0, sizeof(int) * (m + 1));
50 memset(lbuc, 0, sizeof(int) * (m + 1));
51 memset(sbuc, 0, sizeof(int) * (m + 1));
52
53 for (int i = 0; i ≤ n; i++)
| buc[s[i]]++;
54
55
56 for (int i = 1; i ≤ m; i++) {
| buc[i] += buc[i - 1];
57
| lbuc[i] = buc[i - 1];
58 | sbuc[i] = buc[i] - 1;
59 }
60
61
62 tp[n] = s_type;
63 for (int i = n - 1; ~i; i--) {
| if (s[i] < s[i + 1])
| | tp[i] = s_type;
| else if (s[i] > s[i + 1])
| | tp[i] = l_type;
| else
| | tp[i] = tp[i + 1];
64 }
65
66
67
68
69
70
71
72
73 int cnt = 0;
74 for (int i = 1; i ≤ n; i++)
| if (tp[i] == s_type && tp[i - 1] == l_type)
| | pos[cnt++] = i;
75
76
77
78 memset(sa, -1, sizeof(int) * (n + 1));
79 for (int i = 0; i < cnt; i++)
| | sa[sbuc[s[pos[i]]]--] = pos[i];
80 induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
81
82
83 memset(name, -1, sizeof(int) * (n + 1));
84 int lastx = -1, namecnt = 1;
85 bool flag = false;
86
87 for (int i = 1; i ≤ n; i++) {
| | int x = sa[i];
88
| | if (is_lms(tp, x)) {
| | | if (lastx ≥ 0 && !equal_substr(s, x,
| | | | → lastx, tp))
| | | | namecnt++;
| | |
| | | if (lastx ≥ 0 && namecnt == name[lastx])
| | | | flag = true;
| | |
| | | name[x] = namecnt;
| | | lastx = x;
| | }
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     for (int i = 0; i < cnt; i++)
| | tsa[t[i]] = i;
114 }
115 else
116     tsa = sais(t, cnt, namecnt);
117
118 lbuc[0] = sbuc[0] = 0;
119 for (int i = 1; i ≤ m; i++) {
120 | | lbuc[i] = buc[i - 1];
121 | | sbuc[i] = buc[i] - 1;
122 }
123
124
125 memset(sa, -1, sizeof(int) * (n + 1));
126 for (int i = cnt - 1; ~i; i--)
| | sa[sbuc[s[pos[tsa[i]]]]--] = pos[tsa[i]];
127 induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
128
129
130 // 多组数据的时候最好 delete 掉
131 delete[] tp;
132 delete[] pos;
133 delete[] name;
134 delete[] buc;
135 delete[] lbuc;
136 delete[] sbuc;
137 delete[] t;
138 delete[] tsa;
139
140 return sa;
141 }
142
143 // 封装好的函数, 1-based
144 void get_sa(char *s, int n, int *sa, int *rnk, int
| | → height) {
145     static int a[maxn];
146
147     for (int i = 1; i ≤ n; i++)
| | a[i - 1] = s[i];
148
149     a[n] = '$';
150
151     int *t = sais(a, n + 1, 256);
152     memcpy(sa, t, sizeof(int) * (n + 1));
153     delete[] t;
154
155     sa[0] = 0;
156     for (int i = 1; i ≤ n; i++)
| | rnk[+sa[i]] = i;
157
158     for (int i = 1, k = 0; i ≤ n; i++) { // 求 height
159         if (k)
| | | k--;
160
| | | while (s[i + k] == s[sa[rnk[i] - 1] + k])
| | | | k++;
161
162         height[rnk[i]] = k; // height[i] = lcp(sa[i],
| | | | → sa[i - 1])
163     }
164 }
165
166
167
168
169

```

5.3.3 SAMSA

```

1 bool vis[maxn * 2];
2 char s[maxn];
3 int n, id[maxn * 2], ch[maxn * 2][26], height[maxn],
| | → tim = 0;
4
5 void dfs(int x) {

```

```

6   if (id[x]) {
7     height[tim++] = val[last];
8     sa[tim] = id[x];
9
10    last = x;
11  }
12
13  for (int c = 0; c < 26; c++)
14    if (ch[x][c])
15      dfs(ch[x][c]);
16
17  last = par[x];
18}
19
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
41  dfs(1);
42
43  for (int i = 1; i <= n; i++) {
44    if (i > 1)
45      printf(" ");
46    printf("%d", sa[i]); // 1-based
47  }
48  printf("\n");
49
50  for (int i = 1; i < n; i++) {
51    if (i > 1)
52      printf(" ");
53    printf("%d", height[i]);
54  }
55  printf("\n");
56
57  return 0;
}

```

5.4 后缀平衡树

如果不需要查询排名，只需要维护前驱后继关系的题目，可以直接用二分哈希+set去做。

一般的题目需要查询排名，这时候就需要写替罪羊树或者Treap维护tag。插入后缀时如果首字母相同只需比较各自删除首字母后的tag大小即可。

(Treap也具有重量平衡树的性质，每次插入后影响到的子树大小期望是 $O(\log n)$ 的，所以每次做完插入操作之后直接暴力重构子树内tag就行了。)

5.5 后缀自动机

```

1 // 在字符集比较小的时候可以直接开go数组，否则需要用map或
2 // →者哈希表替换
3
4 // 全局变量与数组定义
5 int last, val[maxn], par[maxn], go[maxn][26], sam_cnt;
6 int c[maxn], q[maxn]; // 用来桶排序
7
8 // 在主函数开头加上这句初始化
9 last = sam_cnt = 1;
10
11 // 以下是按val进行桶排序的代码
12 for (int i = 1; i <= sam_cnt; i++) {
13   c[val[i] + 1]++;
14 for (int i = 1; i <= n; i++)
15   c[i] += c[i - 1]; // 这里n是串长
16 for (int i = 1; i <= sam_cnt; i++)
17   q[++c[val[i]]] = i;
18
19 // 加入一个字符 均摊O(1)
20 void extend(int c) {
21   int p = last, np = ++sam_cnt;
22   val[np] = val[p] + 1;
23
24   while (p && !go[p][c]) {
25     go[p][c] = np;
26     p = par[p];
27   }
28
29   if (!p)
30     par[np] = 1;
31   else {
32     int q = go[p][c];
33
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)，但是后者写起来比较麻烦。

```

1 int extend(int p, int c) {
2   int np = 0;
3
4   if (!go[p][c]) {
5     np = ++sam_cnt;
6     val[np] = val[p] + 1;
7     while (p && !go[p][c]) {
8       go[p][c] = np;
9     }
10   }
11
12   last = np;
13 }

```

```

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

```

```

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
18  x → val = tim;
19  x → lazy = true;
20
21
22  update(x → val - val[x → r] + 1, x → val -
23    ↪ val[par[x → l]], 1);
24
25  x → ch[1] = y;
26
27  (y = x) → refresh();
28
29
30  x = x → p;
31
32
33  return y;
34
35
36  // 以下是main函数中的用法
37  for (int i = 1; i ≤ n; i++) {
38    tim++;
39    access(null + id[i]);
40
41    if (i ≥ m) // 例题询问长度是固定的，如果不固定的话就
42      ↪ 按照右端点离线即可
43    ans[i - m + 1] = query(i - m + 1, i);
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
137

```

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。

5.5.2 区间本质不同子串计数(后缀自动机+LCT+线段树)

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

做法：考虑建出后缀自动机，然后枚举右端点，用线段树维护每个左端点的答案

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

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

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

实际上可以发现更新时都是把路径分成若干个整段更新right集合，因此可以用CT维护这个过程。

因此可以用LCA来维护这个过程。
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除了。

```
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);
```

```
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 }
```

```

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));
42
43     return ans;
44 }
45
46 int N;
47
48 void modify(int pos, int u, int v, int &rt) {
49     s = pos;
50     t = u;
51     d = v;
52
53     modify_seg(1, N, rt);
54 }
55
56 int query(int l, int r, int rt) {
57     s = l;
58     t = r;
59     int ans = query_seg(1, N, rt, 0);
60
61     s = 1;
62     t = l;
63     return max(ans, query_seg(1, N, rt, 1) - l);
64 }
65
66 struct node {
67     int size, l, r, id, tim;
68     node *ch[2], *p;
69     bool tag;
70
71     node() = default;
72
73     void apply(int v) {
74         tim = v;
75         tag = true;
76     }
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 →
96         → ch[1];
97 }
98
99 inline bool dir(node *x) {
    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
188             ↪ → r] + 1;
189             modify(left, val[x → r], right + 1,
190                   ↪ root[r]);
191
192         }
193
194         x → apply(r);
195         x → pushdown();
196
197         x → ch[1] = y;
198         (y = x) → update();
199
200         x = x → p;
201     }
202
203     return y;
204 }
205
206 void new_leaf(node *x, node *par) {
207     x → p = par;
208 }
209
210 void new_node(node *x, node *y, node *par) {
211     splay(y);
212
213     if (isroot(y) && y → p == par) {
214         assert(y → ch[0] == null);
215
216         y → ch[0] = x;
217         x → p = y;
218         y → update();
219     }
220     else {
221         splay(par, y);
222
223         assert(y → ch[0] == par);
224         assert(par → ch[1] == null);
225         par → ch[1] = x;
226         x → p = par;
227
228         par → update();
229         y → update();
230
231     }
232
233     x → tim = y → tim;
234 }
235
236 void extend(int c) {
237     int p = sam_last, np = ++sam_cnt;
238     val[np] = val[p] + 1;
239
240     init(null + np, np);
241
242     while (p && !go[p][c]) {
243         go[p][c] = np;
244         p = par[p];
245     }
246
247     if (!p) {
248         par[np] = 1;
249         new_leaf(null + np, null + par[np]);
250     }
251     else {
252         int q = go[p][c];
253
254         if (val[q] == val[p] + 1) {
255             par[np] = q;
256             new_leaf(null + np, null + par[np]);
257         }
258         else {
259             int nq = ++sam_cnt;
260             val[nq] = val[p] + 1;
261             memcpy(go[nq], go[q], sizeof(go[q]));
262
263             init(null + nq, nq);
264
265             new_node(null + nq, null + q, null +
266                      ↪ par[q]);
267             new_leaf(null + np, null + nq);
268
269             par[nq] = par[q];
270             par[np] = par[q] = nq;
271
272             while (p && go[p][c] == q) {
273                 go[p][c] = nq;
274                 p = par[p];
275             }
276
277             sam_last = np;
278         }
279
280     }
281
282     char str[maxn];
283
284     int main() {
285         init(null, 0);
286
287         sam_last = sam_cnt = 1;
288         init(null + 1, 1);
289
290         int n, m;
291         scanf("%s%d", str + 1, &m);
292         n = strlen(str + 1);
293         N = n + m;
294
295         for (int i = 1; i ≤ n; i++) {
296             extend(str[i] - 'a');
297             access(null + sam_last, i);
298
299         }
300
301         int tmp = 0;
302
303         while (m--) {
304             int op;
305             scanf("%d", &op);
306
307             if (op == 1) {
308                 scanf(" %c", &str[++n]);
309             }
310         }
311     }
312 }
```

```

303     str[n] = (str[n] - 'a' + tmp) % 26 + 'a';
304
305     extend(str[n] - 'a');
306     access(null + sam_last, n);
307 }
308 else {
309     int l, r;
310     scanf("%d%d", &l, &r);
311
312     l = (l - 1 + tmp) % n + 1;
313     r = (r - 1 + tmp) % n + 1;
314
315     printf("%d\n", tmp = query(l, r, root[r]));
316 }
317 }
318
319 return 0;
320 }
```

5.6 回文树

```

1 // 定理：一个字符串本质不同的回文子串个数是O(n)的
2 // 注意回文树只需要开一倍结点，另外结点编号也是一个可用
   → 的bfs序
3
4 // 全局数组定义
5 int val[maxn], par[maxn], go[maxn][26], last, cnt;
6 char s[maxn];
7
8 // 重要！在主函数最前面一定要加上以下初始化
9 par[0] = cnt = 1;
10 val[1] = -1;
11 // 这个初始化和广义回文树不一样，写普通题可以用，广义回文
   → 树就不要乱搞了
12
13 // extend函数 均摊O(1)
14 // 向后扩展一个字符
15 // 传入对应下标
16 void extend(int n) {
17     int p = last, c = s[n] - 'a';
18     while (s[n - val[p] - 1] != s[n])
19         p = par[p];
20
21     if (!go[p][c]) {
22         int q = ++cnt, now = p;
23         val[q] = val[p] + 2;
24
25         do
26             p = par[p];
27         while (s[n - val[p] - 1] != s[n]);
28
29         par[q] = go[p][c];
30         last = go[now][c] = q;
31     }
32     else
33         last = go[p][c];
34
35     // a[last]++;
36 }
```

5.6.1 广义回文树

(代码是梯子剖分的版本，压力不大的题目换成直接倍增就好了，常数只差不到一倍)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
```

```

4
5 constexpr int maxn = 1000005, mod = 1000000007;
6
7 int val[maxn], par[maxn], go[maxn][26], fail[maxn][26],
   → pam_last[maxn], pam_cnt;
8 int weight[maxn], pow_26[maxn];
9
10 int trie[maxn][26], trie_cnt, d[maxn], mxd[maxn],
   → son[maxn], top[maxn], len[maxn], sum[maxn];
11 char chr[maxn];
12 int f[25][maxn], log_tbl[maxn];
13 vector<int> v[maxn];
14
15 vector<int> queries[maxn];
16
17 char str[maxn];
18 int n, m, ans[maxn];
19
20 int add(int x, int c) {
21     if (!trie[x][c]) {
22         trie[x][c] = ++trie_cnt;
23         f[0][trie[x][c]] = x;
24         chr[trie[x][c]] = c + 'a';
25     }
26
27     return trie[x][c];
28 }
29
30 int del(int x) {
31     return f[0][x];
32 }
33
34 void dfs1(int x) {
35     mxd[x] = d[x] = d[f[0][x]] + 1;
36
37     for (int i = 0; i < 26; i++)
38         if (trie[x][i])
39             int y = trie[x][i];
40
41             dfs1(y);
42
43             mxd[x] = max(mxd[x], mxd[y]);
44             if (mxd[y] > mxd[son[x]])
45                 son[x] = y;
46
47 }
48
49 void dfs2(int x) {
50     if (x == son[f[0][x]])
51         top[x] = top[f[0][x]];
52     else
53         top[x] = x;
54
55     for (int i = 0; i < 26; i++)
56         if (trie[x][i])
57             int y = trie[x][i];
58             dfs2(y);
59
60
61     if (top[x] == x) {
62         int u = x;
63         while (top[son[u]] == x)
64             u = son[u];
65
66         len[x] = d[u] - d[x];
67
68         for (int i = 0; i < len[x]; i++)
69             v[x].push_back(u);
70             u = f[0][u];
71
72     }
73 }
```

```

74     for (int i = 0; i < len[x]; i++) { // 梯子剖
75         // 分, 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             weight[q] = (weight[par[q]] + (long long)n
131             - val[q] + 1) * pow_26[n - val[q]] % mod;
132         else
133             weight[q] = weight[par[q]];
134
135         pam_last[x] = new_last;
136
137     return weight[pam_last[x]];
138 }
139
140 void bfs() {
141     queue<int> q;
142
143     q.push(1);
144
145     while (!q.empty()) {
146         int x = q.front();
147         q.pop();
148
149         sum[x] = sum[f[0][x]];
150         if (x > 1)
151             sum[x] = (sum[x] + extend(x)) % mod;
152
153         for (int i : queries[x])
154             ans[i] = sum[x];
155
156         for (int i = 0; i < 26; i++)
157             if (trie[x][i])
158                 q.push(trie[x][i]);
159     }
160 }
161
162 int main() {
163
164     pow_26[0] = 1;
165     log_tbl[0] = -1;
166
167     for (int i = 1; i <= 1000000; i++)
168         pow_26[i] = 26ll * pow_26[i - 1] % mod;
169     log_tbl[i] = log_tbl[i / 2] + 1;
170 }
171
172 int T;
173 scanf("%d", &T);
174
175 while (T--) {
176     scanf("%d%d%s", &n, &m, str);
177
178     trie_cnt = 1;
179     chr[1] = '#';
180
181     int last = 1;
182     for (char *c = str; *c; c++)
183         last = add(last, *c - 'a');
184
185     queries[last].push_back(0);
186
187     for (int i = 1; i <= m; i++) {
188         int op;
189         scanf("%d", &op);
190
191         if (op == 1) {
192             char c;
193             scanf(" %c", &c);
194
195             last = add(last, c - 'a');
196         }
197         else
198             last = del(last);
199
200         queries[last].push_back(i);
201     }
202
203     dfs1(1);
204     dfs2(1);
205
206     for (int j = 1; j <= log_tbl[trie_cnt]; j++)
207         for (int i = 1; i <= trie_cnt; i++)
208             f[j][i] = f[j - 1][f[j - 1][i]];
209
210     par[0] = pam_cnt = 1;
211 }

```

```

212
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;
}

```

5.8 字符串原理

KMP和AC自动机的fail指针存储的都是它在串或者字典树上的最长后缀，因此要判断两个前缀是否互为后缀时可以直接用fail指针判断。当然它不能做子串问题，也不能做最长公共后缀。
后缀数组利用的主要是LCP长度可以按照字典序做RMQ的性质，与某个串的LCP长度 \geq 某个值的后缀形成一个区间。另外一个比较好用的性质是本质不同的子串个数 = 所有子串数 - 字典序相邻的串的height。

后缀自动机实际上可以接受的是所有后缀，如果把中间状态也算上的话就是所有子串。它的fail指针代表的也是当前串的后缀，不过注意每个状态可以代表很多状态，只要右端点在right集合中且长度处在 $(val_{par_p}, val_p]$ 中的串都被它代表。

后缀自动机的fail树也就是反串的后缀树。每个结点代表的串和后缀自动机同理，两个串的LCP长度也就是他们在后缀树上的LCA。

5.7 Manacher马拉车

```

1 // n为串长, 回文半径输出到p数组中
2 // 数组要开串长的两倍
3 void manacher(const char *t, int n) {
4     static char s[maxn * 2];
5
6     for (int i = n; i--)
7         s[i * 2] = t[i];
8     for (int i = 0; i ≤ n; i++)
9         s[i * 2 + 1] = '#';
10
11    s[0] = '$';
12    s[(n + 1) * 2] = '\0';
13    n = n * 2 + 1;
14
15    int mx = 0, j = 0;
16
17    for (int i = 1; i ≤ n; i++) {
18        p[i] = (mx > i ? min(p[j * 2 - i], mx - i) :
19            → 1);
20        while (s[i - p[i]] == s[i + p[i]])
21            p[i]++;
22
23        if (i + p[i] > mx) {
24            mx = i + p[i];
25            j = i;
26        }
27    }
}

```

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, 右端
2   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               >= M) ≤ f[i] + calc(i + 1, M)) {
66             L = M + 1;
67           }
68           else
69             R = M;
70         }
71         q[++tail] = i;
72         p[tail] = L;
73       }
74     }
75   }
76 }
77
78 return g[n];
79
80 }
```

```

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   q[++tail] = i;
68   p[tail] = L;
69
70   break;
71 }
72
73 if (head > tail) {
74   q[++tail] = i;
75   p[tail] = i + 1;
76 }
77
78 return g[n];
79
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      int sum = 0;
31      for (int k = i; k < j; k++)
32        sum += a[k];
33      if (sum >= m)
34        ans += (-1)^(j - i - 1) * binom(a[i], j - i - 1);
35    }
36  }
37
38  cout << ans;
39 }
```

```
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所以这个区
32 // → 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
45     dp[0] = 1;
46
47     for (int i = 1; i ≤ n; i++) {
48         dp[i] = (long long)dp[i - 1] * a[i] % p;
49
50         for (int j = 1; j < i; j++) {
51             int tmp = (long long)dp[j - 1] * f[j][i - j
52             → + 1] % p;
53
54             if ((i - j) % 2)
55                 tmp = p - tmp;
56
57             dp[i] = (dp[i] + tmp) % p;
58     }
59
60     printf("%d\n", dp[n]);
61
62     return 0;
63 }
```

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
60 const point inf_point = point(1e18, 1e18);
61

```

```

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

```

```

131     return p.cross(e → origin, e → dest()) > 0;
132 }
133
134 bool right_of(point p, quad_edge *e) {
135     return p.cross(e → origin, e → dest()) < 0;
136 }
137
138 template <class T>
139 T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2,
140        → T c3) {
141     return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 -
142           → c1 * b3) +
143           a3 * (b1 * c2 - c1 * b2);
144 }
145
146 bool in_circle(point a, point b, point c, point d) { // → 如果有__int128就直接计算行列式, 否则算角度
147 #if defined(__LP64__) || defined(__WIN64__)
148     __int128 det = -det3<__int128>(b.x, b.y, b.abs2(),
149           → c.x, c.y, c.abs2(), d.x, d.y, d.abs2());
150     det += det3<__int128>(a.x, a.y, a.abs2(), c.x, c.y,
151           → c.abs2(), d.x, d.y, d.abs2());
152     det -= det3<__int128>(a.x, a.y, a.abs2(), b.x, b.y,
153           → b.abs2(), d.x, d.y, d.abs2());
154     det += det3<__int128>(a.x, a.y, a.abs2(), b.x, b.y,
155           → b.abs2(), c.x, c.y, c.abs2());
156
157     return det > 0;
158 #else
159     auto ang = [] (point l, point mid, point r) {
160         ll x = mid.dot(l, r);
161         ll y = mid.cross(l, r);
162         long double res = atan2((long double)x, (long
163             → double)y);
164         return res;
165     };
166
167     long double kek = ang(a, b, c) + ang(c, d, a) -
168           → ang(b, c, d) - ang(d, a, b);
169
170     return kek > 1e-10;
171 #endif
172 }
173
174 pair<quad_edge*, quad_edge*> divide_and_conquer(int l,
175           → int r, vector<point> &p) {
176     if (r - l + 1 == 2) {
177         quad_edge *res = make_edge(p[l], p[r]);
178         return make_pair(res, res → rev());
179     }
180
181     if (r - l + 1 == 3) {
182         quad_edge *a = make_edge(p[l], p[l + 1]), *b =
183           → make_edge(p[l + 1], p[r]);
184         splice(a → rev(), b);
185
186         int sg = sgn(p[l].cross(p[l + 1], p[r]));
187
188         if (sg == 0)
189             return make_pair(a, b → rev());
190
191         quad_edge *c = connect(b, a);
192
193         if (sg == 1)
194             return make_pair(a, b → rev());
195         else
196             return make_pair(c → rev(), c);
197     }
198
199     int mid = (l + r) / 2;
200
201     quad_edge *ldo, *ldi, *rdo, *rdi;
202     tie(ldo, ldi) = divide_and_conquer(l, mid, p);
203     tie(rdi, rdo) = divide_and_conquer(mid + 1, r, p);
204
205     while (true) {
206         if (left_of(rdi → origin, ldi))
207             ldi = ldi → lnext();
208
209         else if (right_of(ldi → origin, rdi))
210             rdi = rdi → rev() → onext();
211
212         else
213             break;
214     }
215
216     quad_edge *basel = connect(rdi → rev(), ldi);
217     auto is_valid = [&basel] (quad_edge *e) {
218         return right_of(e → dest(), basel);
219     };
220
221     if (ldi → origin == ldo → origin)
222         ldo = basel → rev();
223     if (rdi → origin == rdo → origin)
224         rdo = basel;
225
226     while (true) {
227         quad_edge *lcand = basel → rev() → onext();
228         if (is_valid(lcand)) {
229             while (in_circle(basel → dest(), basel →
230                   → origin, lcand → dest(), lcand → onext(
231                   → dest())))
232                 quad_edge *t = lcand → onext();
233                 delete_edge(lcand);
234                 lcand = t;
235         }
236
237         quad_edge *rcand = basel → oprev();
238         if (is_valid(rcand)) {
239             while (in_circle(basel → dest(), basel →
240                   → origin, rcand → dest(), rcand →
241                   → oprev() → dest()))
242                 quad_edge *t = rcand → oprev();
243                 delete_edge(rcand);
244                 rcand = t;
245         }
246
247         if (!is_valid(lcand) && !is_valid(rcand))
248             break;
249
250         if (!is_valid(lcand) || (is_valid(rcand) &&
251             → in_circle(lcand → dest(), lcand → origin,
252             → rcand → origin, rcand → dest())))
253             basel = connect(rcand, basel → rev());
254         else
255             basel = connect(basel → rev(), lcand →
256               → rev());
257
258         return make_pair(ldo, rdo);
259     }
260
261     vector<tuple<point, point, point> >
262       → delaunay(vector<point> p) { // Delaunay 三角剖分
263         sort(p.begin(), p.end(), [] (const point &a, const
264             → point &b) {

```

```

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

7.2 最近点对

首先分治的做法是众所周知的.

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

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

加入一个新点时只需要枚举自身和周围共计9个方格中的点, 显然枚举到的点最多16个. 如果加入之后答案变小了就 $O(n)$ 暴力重构.

前 i 个点中 i 是最近点对中的点的概率至多为 $\frac{2}{i}$, 所以每个点的期望贡献都是 $O(1)$, 总的复杂度就是期望 $O(n)$.

如果对每个点都要求出距离最近的点的话, 也有随机化的 $O(n)$ 做法:

一个真的随机算法:

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

1. 循环直到删完所有点:

- 随机选一个点, 计算它到所有点的最短距离 d .
 - 将所有点划分到 $l = d/3$ 的网格里, 比如 $(\lfloor \frac{x}{l} \rfloor, \lfloor \frac{y}{l} \rfloor)$.
 - 将九宫格内孤立的点删除, 这意味着这些点的最近点对距离不小于 $\frac{2\sqrt{2}}{3}d$, 其中 $\frac{2\sqrt{2}}{3} < 1$.
2. 取最后一个 d , 将所有点划分到 $(\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor)$ 的网格里, 暴力计算九宫格内的答案.

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

第二部分分析类似分治做法, 周围只有常数个点.

所以总复杂度是 $O(n)$ 的.

8 杂项

8.1 $O(1)$ 快速乘

如果对速度要求很高并且不能用指令集，可以去看fstqwq的模板。

```

1 // long double 快速乘
2 // 在两数直接相乘会爆long long时才有必要使用
3 // 常数比直接long long乘法 + 取模大很多，非必要时不建议使用
4 // → 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(减少浮点数累加的误差)

当然一般来说是用不到的，累加被卡精度了才有必要考虑。

```

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

8.3 Python Decimal

```

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

```

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

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
33    big_decimal(string s) {
34        memset(a, 0, sizeof(a));
35        negative = false;
36
37        if (s == "") {
38            return;
39        }
40
41        if (s[0] == '-') {
42            negative = true;
43            s = s.substr(1);
44        }
45        a[0] = s.size();
46    }
47
48    void print() {
49        cout << a[0];
50
51        for (int i = 1; i < maxn; i++) {
52            cout << a[i];
53        }
54    }
55
56    void add(big_decimal b) {
57        int carry = 0;
58
59        for (int i = 0; i < maxn; i++) {
60            int sum = a[i] + b.a[i] + carry;
61            a[i] = sum % 10;
62            carry = sum / 10;
63        }
64
65        if (carry) {
66            a[++a[0]] = carry;
67        }
68    }
69
70    void sub(big_decimal b) {
71        int borrow = 0;
72
73        for (int i = 0; i < maxn; i++) {
74            int diff = a[i] - b.a[i] - borrow;
75            if (diff < 0) {
76                diff += 10;
77                borrow = 1;
78            } else {
79                borrow = 0;
80            }
81            a[i] = diff;
82        }
83
84        if (borrow) {
85            a[++a[0]] = borrow;
86        }
87    }
88
89    void mult(big_decimal b) {
90        int carry = 0;
91
92        for (int i = 0; i < maxn; i++) {
93            for (int j = 0; j < maxn; j++) {
94                int prod = a[i] * b.a[j] + carry;
95                a[i + j] = prod % 10;
96                carry = prod / 10;
97            }
98        }
99
100       if (carry) {
101           a[++a[0]] = carry;
102       }
103   }
104
105   void divide(big_decimal b) {
106       int quotient = 0;
107
108       for (int i = 0; i < maxn; i++) {
109           quotient = quotient * 10 + a[i];
110
111           int digit = quotient / b.a[0];
112
113           a[i] = digit;
114
115           quotient = quotient % b.a[0];
116       }
117   }
118
119   void mod(big_decimal b) {
120       int remainder = 0;
121
122       for (int i = 0; i < maxn; i++) {
123           remainder = remainder * 10 + a[i];
124
125           int digit = remainder / b.a[0];
126
127           a[i] = digit;
128
129           remainder = remainder % b.a[0];
130       }
131   }
132
133   void pow(int n) {
134       big_decimal result(1);
135
136       for (int i = 0; i < n; i++) {
137           result *= *this;
138       }
139   }
140
141   void log() {
142       double result = 0;
143
144       for (int i = 0; i < maxn; i++) {
145           result += a[i] * log10(10);
146       }
147   }
148
149   void exp() {
150       double result = 1;
151
152       for (int i = 0; i < maxn; i++) {
153           result *= 10;
154       }
155   }
156
157   void ln() {
158       double result = 0;
159
160       for (int i = 0; i < maxn; i++) {
161           result += a[i] * log10(10);
162       }
163   }
164
165   void sqrt() {
166       double result = 1;
167
168       for (int i = 0; i < maxn; i++) {
169           result *= 10;
170       }
171   }
172
173   void log10() {
174       double result = 0;
175
176       for (int i = 0; i < maxn; i++) {
177           result += a[i] * log10(10);
178       }
179   }
180
181   void sin() {
182       double result = 0;
183
184       for (int i = 0; i < maxn; i++) {
185           result += a[i] * sin(10);
186       }
187   }
188
189   void cos() {
190       double result = 0;
191
192       for (int i = 0; i < maxn; i++) {
193           result += a[i] * cos(10);
194       }
195   }
196
197   void tan() {
198       double result = 0;
199
200       for (int i = 0; i < maxn; i++) {
201           result += a[i] * tan(10);
202       }
203   }
204
205   void sinh() {
206       double result = 0;
207
208       for (int i = 0; i < maxn; i++) {
209           result += a[i] * sinh(10);
210       }
211   }
212
213   void cosh() {
214       double result = 0;
215
216       for (int i = 0; i < maxn; i++) {
217           result += a[i] * cosh(10);
218       }
219   }
220
221   void tanh() {
222       double result = 0;
223
224       for (int i = 0; i < maxn; i++) {
225           result += a[i] * tanh(10);
226       }
227   }
228
229   void asin() {
230       double result = 0;
231
232       for (int i = 0; i < maxn; i++) {
233           result += a[i] * asin(10);
234       }
235   }
236
237   void acos() {
238       double result = 0;
239
240       for (int i = 0; i < maxn; i++) {
241           result += a[i] * acos(10);
242       }
243   }
244
245   void atan() {
246       double result = 0;
247
248       for (int i = 0; i < maxn; i++) {
249           result += a[i] * atan(10);
250       }
251   }
252
253   void asinh() {
254       double result = 0;
255
256       for (int i = 0; i < maxn; i++) {
257           result += a[i] * asinh(10);
258       }
259   }
260
261   void acosh() {
262       double result = 0;
263
264       for (int i = 0; i < maxn; i++) {
265           result += a[i] * acosh(10);
266       }
267   }
268
269   void atanh() {
270       double result = 0;
271
272       for (int i = 0; i < maxn; i++) {
273           result += a[i] * atanh(10);
274       }
275   }
276
277   void log2() {
278       double result = 0;
279
280       for (int i = 0; i < maxn; i++) {
281           result += a[i] * log2(10);
282       }
283   }
284
285   void exp2() {
286       double result = 1;
287
288       for (int i = 0; i < maxn; i++) {
289           result *= 10;
290       }
291   }
292
293   void ln2() {
294       double result = 0;
295
296       for (int i = 0; i < maxn; i++) {
297           result += a[i] * ln2(10);
298       }
299   }
300
301   void sqrt2() {
302       double result = 1;
303
304       for (int i = 0; i < maxn; i++) {
305           result *= 10;
306       }
307   }
308
309   void log102() {
310       double result = 0;
311
312       for (int i = 0; i < maxn; i++) {
313           result += a[i] * log102(10);
314       }
315   }
316
317   void exp10() {
318       double result = 1;
319
320       for (int i = 0; i < maxn; i++) {
321           result *= 10;
322       }
323   }
324
325   void ln10() {
326       double result = 0;
327
328       for (int i = 0; i < maxn; i++) {
329           result += a[i] * ln10(10);
330       }
331   }
332
333   void sinh2() {
334       double result = 0;
335
336       for (int i = 0; i < maxn; i++) {
337           result += a[i] * sinh2(10);
338       }
339   }
340
341   void cosh2() {
342       double result = 0;
343
344       for (int i = 0; i < maxn; i++) {
345           result += a[i] * cosh2(10);
346       }
347   }
348
349   void tanh2() {
350       double result = 0;
351
352       for (int i = 0; i < maxn; i++) {
353           result += a[i] * tanh2(10);
354       }
355   }
356
357   void asinh2() {
358       double result = 0;
359
360       for (int i = 0; i < maxn; i++) {
361           result += a[i] * asinh2(10);
362       }
363   }
364
365   void acosh2() {
366       double result = 0;
367
368       for (int i = 0; i < maxn; i++) {
369           result += a[i] * acosh2(10);
370       }
371   }
372
373   void atanh2() {
374       double result = 0;
375
376       for (int i = 0; i < maxn; i++) {
377           result += a[i] * atanh2(10);
378       }
379   }
380
381   void log22() {
382       double result = 0;
383
384       for (int i = 0; i < maxn; i++) {
385           result += a[i] * log22(10);
386       }
387   }
388
389   void exp102() {
390       double result = 1;
391
392       for (int i = 0; i < maxn; i++) {
393           result *= 10;
394       }
395   }
396
397   void ln102() {
398       double result = 0;
399
400       for (int i = 0; i < maxn; i++) {
401           result += a[i] * ln102(10);
402       }
403   }
404
405   void sqrt10() {
406       double result = 1;
407
408       for (int i = 0; i < maxn; i++) {
409           result *= 10;
410       }
411   }
412
413   void log1010() {
414       double result = 0;
415
416       for (int i = 0; i < maxn; i++) {
417           result += a[i] * log1010(10);
418       }
419   }
420
421   void exp1010() {
422       double result = 1;
423
424       for (int i = 0; i < maxn; i++) {
425           result *= 10;
426       }
427   }
428
429   void ln1010() {
430       double result = 0;
431
432       for (int i = 0; i < maxn; i++) {
433           result += a[i] * ln1010(10);
434       }
435   }
436
437   void sin10() {
438       double result = 0;
439
440       for (int i = 0; i < maxn; i++) {
441           result += a[i] * sin10(10);
442       }
443   }
444
445   void cos10() {
446       double result = 0;
447
448       for (int i = 0; i < maxn; i++) {
449           result += a[i] * cos10(10);
450       }
451   }
452
453   void tan10() {
454       double result = 0;
455
456       for (int i = 0; i < maxn; i++) {
457           result += a[i] * tan10(10);
458       }
459   }
460
461   void asinh10() {
462       double result = 0;
463
464       for (int i = 0; i < maxn; i++) {
465           result += a[i] * asinh10(10);
466       }
467   }
468
469   void acosh10() {
470       double result = 0;
471
472       for (int i = 0; i < maxn; i++) {
473           result += a[i] * acosh10(10);
474       }
475   }
476
477   void atanh10() {
478       double result = 0;
479
480       for (int i = 0; i < maxn; i++) {
481           result += a[i] * atanh10(10);
482       }
483   }
484
485   void log210() {
486       double result = 0;
487
488       for (int i = 0; i < maxn; i++) {
489           result += a[i] * log210(10);
490       }
491   }
492
493   void exp1010() {
494       double result = 1;
495
496       for (int i = 0; i < maxn; i++) {
497           result *= 10;
498       }
499   }
500
501   void ln1010() {
502       double result = 0;
503
504       for (int i = 0; i < maxn; i++) {
505           result += a[i] * ln1010(10);
506       }
507   }
508
509   void sqrt1010() {
510       double result = 1;
511
512       for (int i = 0; i < maxn; i++) {
513           result *= 10;
514       }
515   }
516
517   void log21010() {
518       double result = 0;
519
520       for (int i = 0; i < maxn; i++) {
521           result += a[i] * log21010(10);
522       }
523   }
524
525   void exp101010() {
526       double result = 1;
527
528       for (int i = 0; i < maxn; i++) {
529           result *= 10;
530       }
531   }
532
533   void ln101010() {
534       double result = 0;
535
536       for (int i = 0; i < maxn; i++) {
537           result += a[i] * ln101010(10);
538       }
539   }
540
541   void sin1010() {
542       double result = 0;
543
544       for (int i = 0; i < maxn; i++) {
545           result += a[i] * sin1010(10);
546       }
547   }
548
549   void cos1010() {
550       double result = 0;
551
552       for (int i = 0; i < maxn; i++) {
553           result += a[i] * cos1010(10);
554       }
555   }
556
557   void tan1010() {
558       double result = 0;
559
560       for (int i = 0; i < maxn; i++) {
561           result += a[i] * tan1010(10);
562       }
563   }
564
565   void asinh1010() {
566       double result = 0;
567
568       for (int i = 0; i < maxn; i++) {
569           result += a[i] * asinh1010(10);
570       }
571   }
572
573   void acosh1010() {
574       double result = 0;
575
576       for (int i = 0; i < maxn; i++) {
577           result += a[i] * acosh1010(10);
578       }
579   }
580
581   void atanh1010() {
582       double result = 0;
583
584       for (int i = 0; i < maxn; i++) {
585           result += a[i] * atanh1010(10);
586       }
587   }
588
589   void log2101010() {
590       double result = 0;
591
592       for (int i = 0; i < maxn; i++) {
593           result += a[i] * log2101010(10);
594       }
595   }
596
597   void exp10101010() {
598       double result = 1;
599
600       for (int i = 0; i < maxn; i++) {
601           result *= 10;
602       }
603   }
604
605   void ln10101010() {
606       double result = 0;
607
608       for (int i = 0; i < maxn; i++) {
609           result += a[i] * ln10101010(10);
610       }
611   }
612
613   void sin101010() {
614       double result = 0;
615
616       for (int i = 0; i < maxn; i++) {
617           result += a[i] * sin101010(10);
618       }
619   }
620
621   void cos101010() {
622       double result = 0;
623
624       for (int i = 0; i < maxn; i++) {
625           result += a[i] * cos101010(10);
626       }
627   }
628
629   void tan101010() {
630       double result = 0;
631
632       for (int i = 0; i < maxn; i++) {
633           result += a[i] * tan101010(10);
634       }
635   }
636
637   void asinh101010() {
638       double result = 0;
639
640       for (int i = 0; i < maxn; i++) {
641           result += a[i] * asinh101010(10);
642       }
643   }
644
645   void acosh101010() {
646       double result = 0;
647
648       for (int i = 0; i < maxn; i++) {
649           result += a[i] * acosh101010(10);
650       }
651   }
652
653   void atanh101010() {
654       double result = 0;
655
656       for (int i = 0; i < maxn; i++) {
657           result += a[i] * atanh101010(10);
658       }
659   }
660
661   void log210101010() {
662       double result = 0;
663
664       for (int i = 0; i < maxn; i++) {
665           result += a[i] * log210101010(10);
666       }
667   }
668
669   void exp1010101010() {
670       double result = 1;
671
672       for (int i = 0; i < maxn; i++) {
673           result *= 10;
674       }
675   }
676
677   void ln1010101010() {
678       double result = 0;
679
680       for (int i = 0; i < maxn; i++) {
681           result += a[i] * ln1010101010(10);
682       }
683   }
684
685   void sin10101010() {
686       double result = 0;
687
688       for (int i = 0; i < maxn; i++) {
689           result += a[i] * sin10101010(10);
690       }
691   }
692
693   void cos10101010() {
694       double result = 0;
695
696       for (int i = 0; i < maxn; i++) {
697           result += a[i] * cos10101010(10);
698       }
699   }
700
701   void tan10101010() {
702       double result = 0;
703
704       for (int i = 0; i < maxn; i++) {
705           result += a[i] * tan10101010(10);
706       }
707   }
708
709   void asinh10101010() {
710       double result = 0;
711
712       for (int i = 0; i < maxn; i++) {
713           result += a[i] * asinh10101010(10);
714       }
715   }
716
717   void acosh10101010() {
718       double result = 0;
719
720       for (int i = 0; i < maxn; i++) {
721           result += a[i] * acosh10101010(10);
722       }
723   }
724
725   void atanh10101010() {
726       double result = 0;
727
728       for (int i = 0; i < maxn; i++) {
729           result += a[i] * atanh10101010(10);
730       }
731   }
732
733   void log21010101010() {
734       double result = 0;
735
736       for (int i = 0; i < maxn; i++) {
737           result += a[i] * log21010101010(10);
738       }
739   }
740
741   void exp101010101010() {
742       double result = 1;
743
744       for (int i = 0; i < maxn; i++) {
745           result *= 10;
746       }
747   }
748
749   void ln101010101010() {
750       double result = 0;
751
752       for (int i = 0; i < maxn; i++) {
753           result += a[i] * ln101010101010(10);
754       }
755   }
756
757   void sin1010101010() {
758       double result = 0;
759
760       for (int i = 0; i < maxn; i++) {
761           result += a[i] * sin1010101010(10);
762       }
763   }
764
765   void cos1010101010() {
766       double result = 0;
767
768       for (int i = 0; i < maxn; i++) {
769           result += a[i] * cos1010101010(10);
770       }
771   }
772
773   void tan1010101010() {
774       double result = 0;
775
776       for (int i = 0; i < maxn; i++) {
777           result += a[i] * tan1010101010(10);
778       }
779   }
780
781   void asinh1010101010() {
782       double result = 0;
783
784       for (int i = 0; i < maxn; i++) {
785           result += a[i] * asinh1010101010(10);
786       }
787   }
788
789   void acosh1010101010() {
790       double result = 0;
791
792       for (int i = 0; i < maxn; i++) {
793           result += a[i] * acosh1010101010(10);
794       }
795   }
796
797   void atanh1010101010() {
798       double result = 0;
799
800       for (int i = 0; i < maxn; i++) {
801           result += a[i] * atanh1010101010(10);
802       }
803   }
804
805   void log2101010101010() {
806       double result = 0;
807
808       for (int i = 0; i < maxn; i++) {
809           result += a[i] * log2101010101010(10);
810       }
811   }
812
813   void exp10101010101010() {
814       double result = 1;
815
816       for (int i = 0; i < maxn; i++) {
817           result *= 10;
818       }
819   }
820
821   void ln10101010101010() {
822       double result = 0;
823
824       for (int i = 0; i < maxn; i++) {
825           result += a[i] * ln10101010101010(10);
826       }
827   }
828
829   void sin101010101010() {
830       double result = 0;
831
832       for (int i = 0; i < maxn; i++) {
833           result += a[i] * sin101010101010(10);
834       }
835   }
836
837   void cos101010101010() {
838       double result = 0;
839
840       for (int i = 0; i < maxn; i++) {
841           result += a[i] * cos101010101010(10);
842       }
843   }
844
845   void tan101010101010() {
846       double result = 0;
847
848       for (int i = 0; i < maxn; i++) {
849           result += a[i] * tan101010101010(10);
850       }
851   }
852
853   void asinh101010101010() {
854       double result = 0;
855
856       for (int i = 0; i < maxn; i++) {
857           result += a[i] * asinh101010101010(10);
858       }
859   }
860
861   void acosh101010101010() {
862       double result = 0;
863
864       for (int i = 0; i < maxn; i++) {
865           result += a[i] * acosh101010101010(10);
866       }
867   }
868
869   void atanh101010101010() {
870       double result = 0;
871
872       for (int i = 0; i < maxn; i++) {
873           result += a[i] * atanh101010101010(10);
874       }
875   }
876
877   void log210101010101010() {
878       double result = 0;
879
880       for (int i = 0; i < maxn; i++) {
881           result += a[i] * log210101010101010(10);
882       }
883   }
884
885   void exp1010101010101010() {
886       double result = 1;
887
888       for (int i = 0; i < maxn; i++) {
889           result *= 10;
890       }
891   }
892
893   void ln1010101010101010() {
894       double result = 0;
895
896       for (int i = 0; i < maxn; i++) {
897           result += a[i] * ln1010101010101010(10);
898       }
899   }
900
901   void sin10101010101010() {
902       double result = 0;
903
904       for (int i = 0; i < maxn; i++) {
905           result += a[i] * sin10101010101010(10);
906       }
907   }
908
909   void cos10101010101010() {
910       double result = 0;
911
912       for (int i = 0; i < maxn; i++) {
913           result += a[i] * cos10101010101010(10);
914       }
915   }
916
917   void tan10101010101010() {
918       double result = 0;
919
920       for (int i = 0; i < maxn; i++) {
921           result += a[i] * tan10101010101010(10);
922       }
923   }
924
925   void asinh10101010101010() {
926       double result = 0;
927
928       for (int i = 0; i < maxn; i++) {
929           result += a[i] * asinh10101010101010(10);
930       }
931   }
932
933   void acosh10101010101010() {
934       double result = 0;
935
936       for (int i = 0; i < maxn; i++) {
937           result += a[i] * acosh10101010101010(10);
938       }
939   }
940
941   void atanh10101010101010() {
942       double result = 0;
943
944       for (int i = 0; i < maxn; i++) {
945           result += a[i] * atanh10101010101010(10);
946       }
947   }
948
949   void log21010101010101010() {
950       double result = 0;
951
952       for (int i = 0; i < maxn; i++) {
953           result += a[i] * log21010101010101010(10);
954       }
955   }
956
957   void exp101010101010101010() {
958
```

```

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

```

```

172     for (int i = 1; i <= u.a[0] || i <= v.a[0]; i++)
173         o.a[i] += u.a[i] + v.a[i];
174
175         if (o.a[i] >= 10) {
176             o.a[i + 1]++;
177             o.a[i] -= 10;
178         }
179     }
180
181     if (o.a[o.a[0] + 1])
182         o.a[0]++;
183
184     return o;
185 }
186
187 friend big_decimal decimal_minus(const big_decimal
188     &u, const big_decimal &v) { // 保证u, v均为正数
189     // 的话可以直接调用
190     int k = cmp(u, v);
191
192     if (k == -1)
193         return -decimal_minus(v, u);
194     else if (k == 0)
195         return big_decimal(0);
196
197     big_decimal o;
198
199     o.a[0] = u.a[0];
200
201     for (int i = 1; i <= u.a[0]; i++) {
202         o.a[i] += u.a[i] - v.a[i];
203
204         if (o.a[i] < 0) {
205             o.a[i] += 10;
206             o.a[i + 1]--;
207         }
208     }
209
210     while (o.a[0] && !o.a[o.a[0]])
211         o.a[0]--;
212
213     return o;
214 }
215
216 friend big_decimal decimal_multi(const big_decimal
217     &u, const big_decimal &v) {
218     big_decimal o;
219
220     o.a[0] = u.a[0] + v.a[0] - 1;
221
222     for (int i = 1; i <= u.a[0]; i++)
223         for (int j = 1; j <= v.a[0]; j++)
224             o.a[i + j - 1] += u.a[i] * v.a[j];
225
226     for (int i = 1; i <= o.a[0]; i++)
227         if (o.a[i] >= 10) {
228             o.a[i + 1] += o.a[i] / 10;
229             o.a[i] %= 10;
230         }
231
232     if (o.a[o.a[0] + 1])
233         o.a[0]++;
234
235     return o;
236 }
237
238 if (v > u)
239     return make_pair(big_decimal(0), u);
240
241 big_decimal o;
242 o.a[0] = u.a[0] - v.a[0] + 1;
243
244 int m = v.a[0];
245 v <= u.a[0] - m;
246
247 for (int i = u.a[0]; i >= m; i--) {
248     while (u >= v) {
249         u = u - v;
250         o.a[i - m + 1]++;
251     }
252
253     v >= 1;
254 }
255
256 while (o.a[0] && !o.a[o.a[0]])
257     o.a[0]--;
258
259 return make_pair(o, u);
260 }
261
262 friend big_decimal operator + (const big_decimal
263     &u, const big_decimal &v) {
264     if (u.negative || v.negative) {
265         if (u.negative && v.negative)
266             return -decimal_plus(-u, -v);
267
268         if (u.negative)
269             return v - (-u);
270
271         if (v.negative)
272             return u - (-v);
273     }
274
275     return decimal_plus(u, v);
276 }
277
278 friend big_decimal operator - (const big_decimal
279     &u, const big_decimal &v) {
280     if (u.negative || v.negative) {
281         if (u.negative && v.negative)
282             return -decimal_minus(-u, -v);
283
284         if (u.negative)
285             return -decimal_plus(-u, v);
286
287         if (v.negative)
288             return decimal_plus(u, -v);
289     }
290
291     return decimal_minus(u, v);
292 }
293
294 friend big_decimal operator * (const big_decimal
295     &u, const big_decimal &v) {
296     if (u.negative || v.negative) {
297         big_decimal o = decimal_multi(abs(u),
298             abs(v));
299
300         if (u.negative ^ v.negative)
301             return -o;
302         return o;
303     }
304
305     return decimal_multi(u, v);
306 }
307
308 friend pair<big_decimal, big_decimal>
309     decimal_divide(big_decimal u, big_decimal v) {
310     // 整除
311 }
```

```

300 }
301
302 big_decimal operator * (long long x) const {
303     if (x >= 10)
304         return *this * big_decimal(x);
305
306     if (negative)
307         return -(*this * x);
308
309     big_decimal o;
310
311     o.a[0] = a[0];
312
313     for (int i = 1; i <= a[0]; i++)
314         o.a[i] += a[i] * x;
315
316     if (o.a[i] >= 10) {
317         o.a[i + 1] += o.a[i] / 10;
318         o.a[i] %= 10;
319     }
320 }
321
322 if (o.a[a[0] + 1])
323     o.a[0]++;
324
325     return o;
326 }
327
328 friend pair<big_decimal, big_decimal>
329     decimal_div(const big_decimal &u, const
330     big_decimal &v) {
331     if (u.negative || v.negative) {
332         pair<big_decimal, big_decimal> o =
333             decimal_div(abs(u), abs(v));
334
335         if (u.negative ^ v.negative)
336             return make_pair(-o.first, -o.second);
337         return o;
338     }
339
340     return decimal_divide(u, v);
341 }
342
343 friend big_decimal operator / (const big_decimal
344     &u, const big_decimal &v) { // v不能是0
345     if (u.negative || v.negative) {
346         big_decimal o = abs(u) / abs(v);
347
348         if (u.negative ^ v.negative)
349             return -o;
350         return o;
351     }
352
353     return decimal_divide(u, v).first;
354 }
355
356 friend big_decimal operator % (const big_decimal
357     &u, const big_decimal &v) {
358     if (u.negative || v.negative) {
359         big_decimal o = abs(u) % abs(v);
360
361         if (u.negative ^ v.negative)
362             return -o;
363         return o;
364     }
365
366     return decimal_divide(u, v).second;
367 }
368

```

8.5 笛卡尔树

```

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

```

8.6 GarsiaWachs算法($O(n \log n)$ 合并石子)

设序列是 $\{a_i\}$, 从左往右, 找到一个最小的且满足 $a_{k-1} \leq a_{k+1}$ 的 k , 找到后合并 a_k 和 a_{k-1} , 再从当前位置开始向左找最大的 j 满足 $a_j \geq a_k + a_{k-1}$ (当然是指合并前的), 然后把 $a_k + a_{k-1}$ 插到 j 的后面就行. 一直重复, 直到只剩下一堆石子就可以了.

另外在这个过程中, 可以假设 a_1 和 a_n 是正无穷的, 可省略边界的判别. 把 a_0 设为INF, a_{n+1} 设为INF-1, 可实现剩余一堆石子时自动结束.

8.7 常用NTT素数及原根

$p = r \times 2^k + 1$	r	k	最小原根
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
985661441	235	22	3
998244353	119	23	3
1004535809	479	21	3
1005060097*	1917	19	5
2013265921	15	27	31
2281701377	17	27	3
31525197391593473	7	52	3
180143985094819841	5	55	6
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 }
```

8.9 枚举子集

(注意这是 $t \neq 0$ 的写法, 如果可以等于0需要在循环里手动break)

```

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

8.10 STL

8.10.1 vector

- `vector(int nSize)`: 创建一个vector, 元素个数为nSize
- `vector(int nSize, const T &value)`: 创建一个vector, 元素个数为nSize, 且值均为value
- `vector(begin, end)`: 复制[begin, end)区间内另一个数组的元素到vector中
- `void assign(int n, const T &x)`: 设置向量中前n个元素的值为x
- `void assign(const_iterator first, const_iterator last)`: 向量中[first, last)中元素设置成当前向量元素
- `void emplace_back(Args&& ... args)`: 自动构造并push_back一个元素, 例如对一个存储pair的vector可以`v.emplace_back(x, y)`

8.10.2 list

- `assign()` 给list赋值
- `back()` 返回最后一个元素
- `begin()` 返回指向第一个元素的迭代器
- `clear()` 删除所有元素
- `empty()` 如果list是空的则返回true
- `end()` 返回末尾的迭代器
- `erase()` 删除一个元素
- `front()` 返回第一个元素
- `insert()` 插入一个元素到list中
- `max_size()` 返回list能容纳的最大元素数量
- `merge()` 合并两个list
- `pop_back()` 删除最后一个元素
- `pop_front()` 删除第一个元素
- `push_back()` 在list的末尾添加一个元素
- `push_front()` 在list的头部添加一个元素
- `rbegin()` 返回指向第一个元素的逆向迭代器
- `remove()` 从list删除元素
- `remove_if()` 按指定条件删除元素
- `rend()` 指向list末尾的逆向迭代器
- `resize()` 改变list的大小
- `reverse()` 把list的元素倒转
- `size()` 返回list中的元素个数
- `sort()` 给list排序
- `splice()` 合并两个list
- `swap()` 交换两个list
- `unique()` 删除list中重复的元

8.10.3 unordered_set/map

- `unordered_map<int, int, hash>`: 自定义哈希函数, 其中hash是一个带重载括号的类.

8.11 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>,
   → pairing_heap_tag> pq;
```

效率参考:

- * 共有五种操作: push、pop、modify、erase、join
- * pairing_heap_tag: push和join为 $O(1)$, 其余为均摊 $\Theta(\log n)$
- * binary_heap_tag: 只支持push和pop, 均为均摊 $\Theta(\log n)$
- * binomial_heap_tag: push为均摊 $O(1)$, 其余为 $\Theta(\log n)$
- * rc_binomial_heap_tag: push为 $O(1)$, 其余为 $\Theta(\log n)$
- * thin_heap_tag: push为 $O(1)$, 不支持join, 其余为 $\Theta(\log n)$; 果只有increase_key, 那么modify为均摊 $O(1)$
- * “不支持”不是不能用, 而是用起来很慢 csdn.net/TRiddle

常用操作:

- `push()`: 向堆中压入一个元素, 返回迭代器
- `pop()`: 将堆顶元素弹出
- `top()`: 返回堆顶元素
- `size()`: 返回元素个数
- `empty()`: 返回是否非空
- `modify(point_iterator, const key)`: 把迭代器位置的key修改为传入的key
- `erase(point_iterator)`: 把迭代器位置的键值从堆中删除
- `join(__gnu_pbds::priority_queue &other)`: 把other合并到*`this`, 并把other清空

8.11.3 平衡树

```

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,
   → tree_order_statistics_node_update> t;
6
7 // rb_tree_tag 红黑树(还有splay_tree_tag和ov_tree_tag,
   → 后者不知道是什么)
```

注意第五个参数要填tree_order_statistics_node_update才能使用排名操作.

- `insert(x)`: 向树中插入一个元素x, 返回`pair<point_iterator, bool>`
 - `erase(x)`: 从树中删除一个元素/迭代器x, 返回一个`bool`表明是否删除成功
 - `order_of_key(x)`: 返回x的排名, 0-based
 - `find_by_order(x)`: 返回排名(0-based)所对应元素的迭代器
 - `lower_bound(x) / upper_bound(x)`: 返回第一个 \geq 或者 $>$ x的元素的迭代器
 - `join(x)`: 将x树并入当前树, 前提是两棵树的类型一样, 并且二者值域不能重叠, x树会被删除
 - `split(x, b)`: 分裂成两部分, 小于等于x的属于当前树, 其余的属于b树
 - `empty()`: 返回是否为空
 - `size()`: 返回大小

(注意平衡树不支持多重值，如果需要多重值，可以再开一个unordered_map来记录值出现的次数，将x<<32后加上出现的次数后插入。注意此时应该为long long类型。)

8.12 rope

```
1 #include <ext/rope>
2 using namespace __gnu_cxx;
3
4 push_back(x); // 在末尾添加x
5 insert(pos, x); // 在pos插入x, 自然支持整个char数组的一
   ↳ 次插入
6 erase(pos, x); // 从pos开始删除x个, 不要只传一个参数, 有
   ↳ 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, 或者说到原点的欧几里得距离.

8.13.2 <algorithm>

- `std::all_of(begin, end, f)`: 检查范围内元素调用函数f后是否全返回真. 类似地还有`std::any_of`和`std::none_of`.
 - `std::for_each(begin, end, f)`: 对范围内所有元素调用一次f. 如果传入的是引用, 也可以用f修改. (例如`for_each(a, a + n, [](int &x){cout << ++x << "\n";})`)
 - `std::for_each_n(begin, n, f)`: 同上, 只不过范围改成了从begin开始的n个元素.
 - `std::copy()`, `std::copy_n()`: 用法谁都会, 但标准里说如果元素是可平凡复制的(比如int), 那么它会避免批量赋值, 并且调用`std::memmove()`之类的快速复制函数. (一句话总结: 它跑得快)
 - `std::rotate(begin, mid, end)`: 循环移动, 移动后mid位置的元素会跑到first位置. C++11起会返回begin位置的元素移动后的位置.
 - `std::unique(begin, end)`: 去重, 返回去重后的end.
 - `std::partition(begin, end, f)`: 把f为true的放在前面, false的放在后面, 返回值是第二部分的开头, 不保持相对顺序. 如果要保留相对顺序可以用`std::stable_partition()`, 比如写整体二分.

- `std::partition_copy(begin, end, begin_t, begin_f, f)`: 不修改原数组, 把true的扔到begin_t, false的扔到begin_f. 返回值是两部分结尾的迭代器的pair.
 - `std::equal_range(begin, end, x)`: 在已经排好序的数组里找到等于x的范围.
 - `std::minmax(a, b)`: 返回pair(`min(a, b)`, `max(a, b)`). 但是要注意返回的是引用, 所以不能直接用来交换 l, r .

8.13.3 std::tuple

- `std::make_tuple(...)`: 返回构造好的tuple
 - `std::get<i>(tup)`: 返回tuple的第*i*项
 - `std::tuple_cat(...)`: 传入几个tuple, 返回按顺序连起来的tuple
 - `std::tie(x, y, z, ...)`: 把传入的变量的左值引用绑起来作为tuple返回, 例如可以`std::tie(x, y, z) = std::make_tuple(a, b, c)`.

8.13.4 <complex>

- `complex<double> imaginary = 1i, x = 2 + 3i;` 可以这样直接构造复数.
 - `real/imag(x)`: 返回实部/虚部.
 - `conj(x)`: 返回共轭复数.
 - `arg(x)`: 返回辐角.
 - `norm(x)`: 返回模的平方. (直接求模用`abs(x)`.)
 - `polar(len, theta)`: 用绝对值和辐角构造复数.

8.14 一些游戏

8.14.1 德州扑克

一般来说德扑里Ace都是最大的，所以把Ace的点数规定为14会好写许多。
附一个高低奥马哈的参考代码，除了有四张底牌和需要比低之外和德扑区别不大。

```
1 struct Card {
2     int suit, value; // Ace is treated as 14
3
4     Card(string s) {
5         char a = s[0];
6
7         if (isdigit(a))
8             value = a - '0';
9         else if (a == 'T')
10            value = 10;
11        else if (a == 'A')
12            value = 14;
13        else if (a == 'J')
14            value = 11;
15        else if (a == 'Q')
16            value = 12;
17        else if (a == 'K')
18            value = 13;
19        else
20            value = -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) {
```

```

31     |     return a.value == b.value;
32   }
33 };
34
35 constexpr int Highcard = 1, Pair = 2, TwoPairs = 3,
36   ↪ ThreeofaKind = 4, Straight = 5,
37 Flush = 6, FullHouse = 7, FourofaKind = 8,
38   ↪ StraightFlush = 9;
39
40 struct Hand {
41     vector<Card> v;
42     int type;
43
44     Hand() : type(0) {}
45
46     Hand(const Hand &o) : v(o.v), type(o.type) {}
47
48     Hand(const vector<Card> &v) : v(v), type(0) {}
49
50     void init_high() {
51         sort(v.begin(), v.end()); // 升序排序
52
53         bool straight = false;
54         if (v.back().value == 14) {
55             if (v[0].value == 2 && v[1].value == 3 &&
56               ↪ v[2].value == 4 && v[3].value == 5) {
57                 straight = true;
58                 rotate(v.begin(), v.begin() + 1,
59                   ↪ v.end());
60             }
61
62             if (!straight) {
63                 bool ok = true;
64                 for (int i = 1; i < 5; i++)
65                     ok &= (v[i].value == v[i - 1].value +
66                           ↪ 1);
67
68                 if (ok)
69                     straight = true;
70             }
71
72             bool flush = all_of(v.begin(), v.end(), [&]
73               ↪ (const Card &a) {return a.suit ==
74                 ↪ v.front().suit;});
75
76             if (flush && straight) { // 同花顺
77                 type = StraightFlush;
78                 reverse(v.begin(), v.end());
79                 return;
80             }
81
82             vector<int> c;
83             c.assign(15, 0);
84
85             for (auto &o : v)
86                 c[o.value]++;
87
88             vector<int> kind[5];
89
90             for (int i = 2; i ≤ 14; i++)
91                 if (c[i] > 1)
92                     kind[c[i]].push_back(i);
93
94             if (!kind[4].empty()) { // 四条
95                 type = FourofaKind;
96
97                 for (int i = 0; i < 4; i++)
98                     if (v[i].value != kind[4].front())
99                         swap(v[i], v.back());
100
101             }
102
103             sort(v.begin(), v.end(), [&] (const Card
104               ↪ &a, const Card &b) {
105                 bool ta = (a.value == kind[3].front()),
106                   ↪ tb = (b.value == kind[3].front());
107
108                 return ta > tb;
109             });
110
111             return;
112         }
113
114         if (flush) {
115             type = Flush;
116             sort(v.begin(), v.end());
117             reverse(v.begin(), v.end());
118
119             return;
120
121         if (straight) {
122             type = Straight;
123             reverse(v.begin(), v.end());
124             return;
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             return;
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                 return a.value > b.value;
154             });
155
156             return;
157
158         if ((int)kind[2].size() == 1) {
159
160             break;
161         }
162
163         return;
164
165     }
166
167     return;
168
169 }
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193

```

```

94     |     break;
95   }
96
97   return;
98 }
99
100 if (!kind[3].empty() && !kind[2].empty()) {
101     type = FullHouse;
102
103     sort(v.begin(), v.end(), [&] (const Card
104       ↪ &a, const Card &b) {
105           bool ta = (a.value == kind[3].front()),
106             ↪ tb = (b.value == kind[3].front());
107
108           return ta > tb;
109     });
110
111     return;
112
113     if (flush) {
114         type = Flush;
115         sort(v.begin(), v.end());
116         reverse(v.begin(), v.end());
117
118         return;
119     }
120
121     if (straight) {
122         type = Straight;
123         reverse(v.begin(), v.end());
124         return;
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         return;
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             return a.value > b.value;
154         });
155
156         return;
157
158     if ((int)kind[2].size() == 1) {
159
160         break;
161     }
162
163     return;
164
165 }
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193

```

```

157     type = Pair;
158
159     sort(v.begin(), v.end(), [&] (const Card
160         &a, const Card &b) {
161         bool ta = (c[a.value] == 2), tb =
162             (c[b.value] == 2);
163
164         if (ta != tb)
165             return ta > tb;
166
167         return a.value > b.value;
168     });
169
170     return;
171 }
172
173 type = Highcard;
174
175 sort(v.begin(), v.end());
176 reverse(v.begin(), v.end());
177
178 void init_low() {
179     for (auto &o : v)
180         if (o.value == 14)
181             o.value = 1;
182
183 sort(v.begin(), v.end());
184 reverse(v.begin(), v.end());
185 }
186
187 friend int cmp_high(const Hand &a, const Hand &b) {
188     if (a.type != b.type)
189         return a.type < b.type ? -1 : 1;
190
191     if (a.v != b.v)
192         return a.v < b.v ? -1 : 1;
193
194     return 0;
195 }
196
197 friend bool small_high(const Hand &a, const Hand
198     &b) {
199     return cmp_high(a, b) < 0;
200 }
201
202 friend int cmp_low(const Hand &a, const Hand &b) {
203     for (int i = 0; i < 5; i++)
204         if (a.v[i].value != b.v[i].value)
205             return a.v[i] < b.v[i] ? 1 : -1;
206
207     return 0;
208 }
209
210 friend bool small_low(const Hand &a, const Hand &b) {
211     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     }
225 }
226 Hand get_max_high(vector<Card> u, vector<Card> v) { // // →private, public
227     Hand ans;
228
229     for (int i = 0; i < 4; i++)
230         for (int j = i + 1; j < 4; j++)
231             for (int k = 0; k < 5; k++)
232                 for (int p = k + 1; p < 5; p++)
233                     for (int q = p + 1; q < 5; q++) {
234                         Hand tmp{u[i], u[j], v[k],
235                             &v[p], &v[q]};;
236
237                         tmp.init_high();
238
239                         if (!ans || cmp_high(tmp, ans) > 0)
240                             ans = tmp;
241
242     return ans;
243 }
244
245 Hand get_max_low(vector<Card> tu, vector<Card> tv) {
246     vector<Card> u, v;
247
248     for (auto o : tu)
249         if (o.value == 14 || o.value <= 8)
250             u.push_back(o);
251
252     for (auto o : tv)
253         if (o.value == 14 || o.value <= 8)
254             v.push_back(o);
255
256     Hand ans;
257
258     for (int i = 0; i < (int)u.size(); i++)
259         for (int j = i + 1; j < (int)u.size(); j++)
260             for (int k = 0; k < (int)v.size(); k++)
261                 for (int p = k + 1; p < (int)v.size();
262                     p++)
263                     for (int q = p + 1; q < (int)v.size(); q++) {
264                         vector<Card> vec = {u[i], u[j],
265                             &v[k], &v[p], &v[q]};
266
267                         bool bad = false;
268
269                         for (int a = 0; a < 5; a++)
270                             for (int b = a + 1; b < 5;
271                                 b++)
272                                 if (vec[a].value ==
273                                     vec[b].value)
274                                     bad = true;
275
276                         if (bad)
277                             continue;
278
279                         Hand tmp(vec);
280
281                         tmp.init_low();
282
283                         if (!ans || cmp_low(tmp, ans) > 0)
284                             ans = tmp;
285
286     }
287 }
```

```

283     return ans;
284 }
285
286 int main() {
287     ios::sync_with_stdio(false);
288
289     int T;
290     cin >> T;
291
292     while (T--) {
293         int p;
294         cin >> p;
295
296         vector<Card> alice, bob, pub;
297
298         for (int i = 0; i < 4; i++) {
299             string s;
300             cin >> s;
301             alice.push_back(Card(s));
302         }
303
304         for (int i = 0; i < 4; i++) {
305             string s;
306             cin >> s;
307             bob.push_back(Card(s));
308         }
309
310         for (int i = 0; i < 5; i++) {
311             string s;
312             cin >> s;
313             pub.push_back(Card(s));
314         }
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             cout << ans[0] << ' ' << ans[1] << '\n';
361
362         }
363     }
364 }
```

```

350
351     if (!dh) {
352         ans[0] += p - p / 2 - (p - p / 2) / 2;
353         ans[1] += (p - p / 2) / 2;
354     }
355     else
356         ans[dh == -1] += p - p / 2;
357
358     cout << ans[0] << ' ' << ans[1] << '\n';
359
360 }
361
362 return 0;
363 }
```

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1.9.7.卡特兰数, 施罗德数, 默慈金数(16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1.9.7.卡特兰数, 施罗德数, 默慈金数(16页).

3. 小施罗德数(A001003)

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859, 2646723, 13648869, 71039373, 372693519, ... (0-based)

性质位置同上.

小施罗德数除了第0项以外都是施罗德数的一半.

4. 默慈金数(Motzkin numbers, A001006)

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634, 310572, 853467, 2356779, ... (0-based)

性质位置同上.

5. 将点按顺序排成一圈后不自交的树的个数(A001764)

1, 1, 3, 12, 55, 273, 1428, 7752, 43263, 246675, 1430715, 8414640, 50067108, 300830572, 1822766520, ... (0-based)

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

也就是说, 在圆上按顺序排列的 n 个点之间连 $n - 1$ 条不相交(除端点外)的弦, 组成一棵树的方案数.

也等于每次只能向右或向上, 并且不能高于 $y = 2x$ 这条直线, 从 $(0, 0)$ 走到 $(n, 2n)$ 的方案数.

扩展: 如果改成不能高于 $y = kx$ 这条直线, 走到 (n, kn) 的方案数, 那么答案就是 $\frac{\binom{(k+1)n}{n}}{kn+1}$.

6. n 个点的圆上画不相交的弦的方案数(A054726)

1, 1, 2, 8, 48, 352, 2880, 25216, 231168, 2190848, 21292032, 211044352, 2125246464, 21681954816, ... (0-based)

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

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, 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1.9.2.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++17`: 狗都知道

- `-Wall -Wextra -Wshadow -Wconversion`: 更多警告

– `-Werror`: 强制将所有Warning变成Error

- `-fsanitize=address/undefined`: 检查有符号整数溢出(算ub)/数组越界

– 注意无符号类型溢出不算ub.

- `-fno-ms-extensions`: 关闭一些和msvc保持一致的特性, 例如, 不标返回值类型的函数会报CE而不是默认为int.

– 但是不写return的话它还是管不了.

8.17 附录: VScode相关

8.17.1 插件

- Chinese (Simplified) (简体中文语言包)
- C/C++
- C++ Intellisense (前提是让用)
- Better C++ Syntax
- Python
- Pylance (前提是让用)
- Rainbow Brackets (前提是让用)

8.17.2 设置选项

- Editor: Insert Spaces (取消勾选, 改为tab缩进)
- Editor: Line Warp (开启折行)
- 改配色, “深色+：默认深色”
- 自动保存(F1 → “auto”)
- Terminal → Integrated: Cursor Style (修改终端光标形状)
- Terminal → Integrated: Cursor Blinking (终端光标闪烁)
- 字体改为Cascadia Code/Mono SemiLight (Windows可用)

8.17.3 快捷键

- F1 / Ctrl+Shift+P: 万能键, 打开命令面板
- F8: 下一个Error Shift+F8: 上一个Error
- Ctrl+\: 水平分栏, 最多3栏
- Ctrl+1/2/3: 切到对应栏
- Ctrl+[/]: 当前行向左/右缩进
- Alt+F12: 查看定义的缩略图(显示小窗, 不跳过去)
- Ctrl+H: 查找替换
- Ctrl+D: 下一个匹配的也被选中(用于配合Ctrl+F)
- Ctrl+U: 回退上一个光标操作(防止光标飞了找不回去)
- Ctrl+/: 切换行注释
- Ctrl+‘(键盘左上角的倒引号): 显示终端

更多快捷键参见最后两页, 分别是Windows和Linux下的快捷键列表。

8.18 附录：骂人的艺术—梁实秋

古今中外没有一个不骂人的人。骂人就是有道德观念的意思, 因为在骂人的时候, 至少在骂人者自己总觉得那人有该骂的地方。何者该骂, 何者不该骂, 这个抉择的标准, 是极道德的。所以根本不骂人, 大可不必。骂人是一种发泄感情的方法, 尤其是那一种怒怒的感情。想骂人的时候而不骂, 时常在身体上弄出毛病, 所以想骂人时, 骂骂何妨?

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

1. 知己知彼

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

2. 无骂不如己者

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

3. 适可而止

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

4. 旁敲侧击

他偷东西, 你骂他是贼; 他抢东西, 你骂他是盗, 这是笨伯。骂人必须先明虚实掩映之法, 须要烘托旁衬, 旁敲侧击, 于要紧处只一语

便得, 所谓杀人于咽喉处着刀。越要骂他你越要原谅他, 即便说些恭维话亦不为过, 这样的骂法才能显得你所骂的句句是真实确凿, 让旁人看起来也可见得你的度量。

5. 态度镇定

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

6. 出言典雅

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

7. 以退为进

两人对骂, 而自己亦有理屈之处, 则处于开骂伊始, 特宜注意, 最好是毅然将自己理屈之处完全承认下来, 即使道歉认错均不妨事。先把自己理屈之处轻轻遮掩过去, 然后你再重整旗鼓, 着着逼人, 方可无后顾之忧。即使自己没有理屈的地方, 也绝不可自行夸张, 务必要谦逊不遑, 把自己的位置降到一个不可再降的位置, 然后骂起人来, 自有一种公正光明的态度。否则你骂他一两句, 他便以你个人的事反唇相讥, 一场对骂, 会变成两人私下口角, 是非曲直, 无从判断。所以骂人者自己要低声下气, 此所谓以退为进。

8. 预设埋伏

你把这句话骂过去, 你便要想看, 他将用什么话骂回来。有眼光的骂人者, 便处处留神, 或是先将他要骂你的话替他说出来, 或是预先安设埋伏, 令他骂回来的话失去效力。他骂你的话, 你替他说出来, 这便等于缴了他的械一般。预设埋伏, 便是在要攻击你的地方, 你先轻轻的安下话根, 然后他骂过来就等于枪弹打在沙包上, 不能中伤。

9. 小题大做

如对方有该骂之处, 而题目身小, 不值一骂, 或你所知不多, 不足一骂, 那时节你便可用小题大做的方法, 来扩大题目。先用诚恳而怀疑的态度引申对方的意思, 由不紧要之点引到大题目上去, 处处用严谨的逻辑逼他说出不逻辑的话来, 或是逼他说出合于逻辑但不合乎理的话来, 然后你再大举骂他, 骂到体无完肤为止, 而原来惹动你的小题目, 轻轻一提便了。

10. 远交近攻

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

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

8.19 附录: Cheat Sheet

见后面几页。

Theoretical Computer Science Cheat Sheet

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

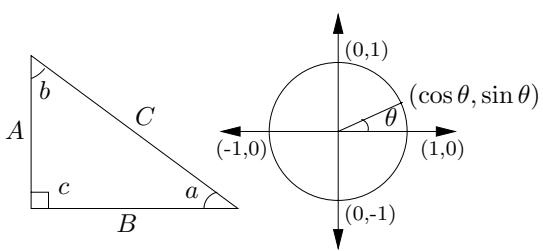
Theoretical Computer Science Cheat Sheet		
Identities Cont.		Trees
38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix}$,	39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+k}{2n}$,	Every tree with n vertices has $n-1$ edges.
40. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} (-1)^{n-k}$,	41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k}$,	Kraft inequality: If the depths of the leaves of a binary tree are d_1, \dots, d_n : $\sum_{i=1}^n 2^{-d_i} \leq 1,$
42. $\begin{Bmatrix} m+n+1 \\ m \end{Bmatrix} = \sum_{k=0}^m k \begin{Bmatrix} n+k \\ k \end{Bmatrix}$,	43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^m k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix}$,	and equality holds only if every internal node has 2 sons.
44. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \binom{k}{m} (-1)^{m-k}$,	45. $(n-m)! \begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^{m-k}$, for $n \geq m$,	
46. $\begin{Bmatrix} n \\ n-m \end{Bmatrix} = \sum_k \begin{bmatrix} m-n \\ m+k \end{bmatrix} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix}$,	47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \begin{bmatrix} m-n \\ m+k \end{bmatrix} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix}$,	
48. $\begin{Bmatrix} n \\ \ell+m \end{Bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k}$,	49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}$.	
Recurrences		
<p>Master method: $T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$</p> <p>If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$.</p> <p>If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.</p> <p>If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then $T(n) = \Theta(f(n))$.</p> <p>Substitution (example): Consider the following recurrence $T_{i+1} = 2^{2^i} \cdot T_i^2$, $T_1 = 2$.</p> <p>Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have $t_{i+1} = 2^i + 2t_i$, $t_1 = 1$.</p> <p>Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get $\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}$.</p> <p>Substituting we find $u_{i+1} = \frac{1}{2} + u_i$, $u_1 = \frac{1}{2}$,</p> <p>which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.</p> <p>Summing factors (example): Consider the following recurrence $T(n) = 3T(n/2) + n$, $T(1) = 1$.</p> <p>Rewrite so that all terms involving T are on the left side $T(n) - 3T(n/2) = n$.</p> <p>Now expand the recurrence, and choose a factor which makes the left side “telescope”</p>	$1(T(n) - 3T(n/2) = n)$ $3(T(n/2) - 3T(n/4) = n/2)$ $\vdots \quad \vdots \quad \vdots$ $3^{\log_2 n-1}(T(2) - 3T(1) = 2)$ <p>Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get</p> $\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$ <p>Let $c = \frac{3}{2}$. Then we have</p> $\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$ <p>and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider</p> $T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$ <p>Note that</p> $T_{i+1} = 1 + \sum_{j=0}^i T_j.$ <p>Subtracting we find</p> $\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$ <p>And so $T_{i+1} = 2T_i = 2^{i+1}$.</p>	<p>Generating functions:</p> <ol style="list-style-type: none"> Multiply both sides of the equation by x^i. Sum both sides over all i for which the equation is valid. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$. Rewrite the equation in terms of the generating function $G(x)$. Solve for $G(x)$. The coefficient of x^i in $G(x)$ is g_i. <p>Example: $g_{i+1} = 2g_i + 1$, $g_0 = 0$.</p> <p>Multiply and sum: $\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i$.</p> <p>We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:</p> $\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$ <p>Simplify: $\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$</p> <p>Solve for $G(x)$:</p> $G(x) = \frac{x}{(1-x)(1-2x)}.$ <p>Expand this using partial fractions:</p> $\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1)x^{i+1}. \end{aligned}$ <p>So $g_i = 2^i - 1$.</p>

Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159, e \approx 2.71828, \gamma \approx 0.57721, \phi = \frac{1+\sqrt{5}}{2} \approx 1.61803, \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$				
i	2^i	p_i	General	Probability
1	2	2	Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$): $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},$ $B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$	Continuous distributions: If $\Pr[a < X < b] = \int_a^b p(x) dx,$ then p is the probability density function of X . If $\Pr[X < a] = P(a),$
2	4	3		then P is the distribution function of X . If P and p both exist then $P(a) = \int_{-\infty}^a p(x) dx.$
3	8	5	Change of base, quadratic formula: $\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$	Expectation: If X is discrete $E[g(X)] = \sum_x g(x) \Pr[X = x].$
4	16	7	Euler's number e :	If X continuous then $E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$
5	32	11	$e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$	Variance, standard deviation: $\text{VAR}[X] = E[X^2] - E[X]^2,$ $\sigma = \sqrt{\text{VAR}[X]}.$
6	64	13	$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$	For events A and B : $\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$
7	128	17	$(1 + \frac{1}{n})^n < e < (1 + \frac{1}{n})^{n+1}.$	$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$ iff A and B are independent.
8	256	19	$(1 + \frac{1}{n})^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$	$\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$
9	512	23	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.
10	1,024	29		$E[X + Y] = E[X] + E[Y],$ $E[cX] = cE[X].$
11	2,048	31		Bayes' theorem:
12	4,096	37		$\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$
13	8,192	41		Inclusion-exclusion:
14	16,384	43		$\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] + \sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$
15	32,768	47		Moment inequalities:
16	65,536	53	$\ln n < H_n < \ln n + 1,$	$\Pr[X \geq \lambda E[X]] \leq \frac{1}{\lambda},$
17	131,072	59	$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$	$\Pr[X - E[X] \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$
18	262,144	61	Factorial, Stirling's approximation: $1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$	Geometric distribution: $\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$
19	524,288	67		$E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$
20	1,048,576	71		
21	2,097,152	73		
22	4,194,304	79		
23	8,388,608	83		
24	16,777,216	89	Ackermann's function and inverse: $a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$	
25	33,554,432	97		
26	67,108,864	101		
27	134,217,728	103	$\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$	
28	268,435,456	107	Binomial distribution: $\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$	
29	536,870,912	109	$E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$	
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:

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

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

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

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

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

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

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

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

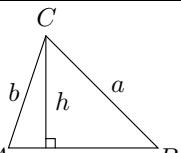
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$

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

θ	$\sin \theta$	$\cos \theta$	$\tan \theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

... in mathematics you don't understand things, you just get used to them.
– J. von Neumann

More Trig.



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

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

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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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> $C \equiv r_1 \pmod{m_1}$ $\vdots \vdots \vdots$ $C \equiv r_n \pmod{m_n}$ <p>if m_i and m_j are relatively prime for $i \neq j$.</p> <p>Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then</p> $\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$ <p>Euler's theorem: If a and b are relatively prime then</p> $1 \equiv a^{\phi(b)} \pmod{b}.$ <p>Fermat's theorem:</p> $1 \equiv a^{p-1} \pmod{p}.$ <p>The Euclidean algorithm: if $a > b$ are integers then</p> $\gcd(a, b) = \gcd(a \bmod b, b).$ <p>If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then</p> $S(x) = \sum_{d x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$ <p>Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.</p> <p>Wilson's theorem: n is a prime iff</p> $(n-1)! \equiv -1 \pmod{n}.$ <p>Möbius inversion:</p> $\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$ <p>If</p> $G(a) = \sum_{d a} F(d),$ <p>then</p> $F(a) = \sum_{d a} \mu(d) G\left(\frac{a}{d}\right).$ <p>Prime numbers:</p> $p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$ $\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$	<p>Definitions:</p> <ul style="list-style-type: none"> <i>Loop</i>: An edge connecting a vertex to itself. <i>Directed</i>: Each edge has a direction. <i>Simple</i>: Graph with no loops or multi-edges. <i>Walk</i>: A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$. <i>Trail</i>: A walk with distinct edges. <i>Path</i>: A trail with distinct vertices. <i>Connected</i>: A graph where there exists a path between any two vertices. <i>Component</i>: A maximal connected subgraph. <i>Tree</i>: A connected acyclic graph. <i>Free tree</i>: A tree with no root. <i>DAG</i>: Directed acyclic graph. <i>Eulerian</i>: Graph with a trail visiting each edge exactly once. <i>Hamiltonian</i>: Graph with a cycle visiting each vertex exactly once. <i>Cut</i>: A set of edges whose removal increases the number of components. <i>Cut-set</i>: A minimal cut. <i>Cut edge</i>: A size 1 cut. <i>k-Connected</i>: A graph connected with the removal of any $k-1$ vertices. <i>k-Tough</i>: $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G-S) \leq S$. <i>k-Regular</i>: A graph where all vertices have degree k. <i>k-Factor</i>: A k-regular spanning subgraph. <i>Matching</i>: A set of edges, no two of which are adjacent. <i>Clique</i>: A set of vertices, all of which are adjacent. <i>Ind. set</i>: A set of vertices, none of which are adjacent. <i>Vertex cover</i>: A set of vertices which cover all edges. <i>Planar graph</i>: A graph which can be embedded in the plane. <i>Plane graph</i>: An embedding of a planar graph. <p style="text-align: right;">$\sum_{v \in V} \deg(v) = 2m.$</p>	<p>Notation:</p> <ul style="list-style-type: none"> $E(G)$: Edge set $V(G)$: Vertex set $c(G)$: Number of components $G[S]$: Induced subgraph $\deg(v)$: Degree of v $\Delta(G)$: Maximum degree $\delta(G)$: Minimum degree $\chi(G)$: Chromatic number $\chi_E(G)$: Edge chromatic number G^c: Complement graph K_n: Complete graph K_{n_1, n_2}: Complete bipartite graph $r(k, \ell)$: Ramsey number 							
		Geometry							
<p>Projective coordinates: triples (x, y, z), not all x, y and z zero.</p> $(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$ <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">Cartesian</td><td style="width: 50%;">Projective</td></tr> <tr> <td>(x, y)</td><td>$(x, y, 1)$</td></tr> <tr> <td>$y = mx + b$</td><td>$(m, -1, b)$</td></tr> <tr> <td>$x = c$</td><td>$(1, 0, -c)$</td></tr> </table> <p>Distance formula, L_p and L_∞ metric:</p> $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$ $[x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p},$ $\lim_{p \rightarrow \infty} [x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p}.$ <p>Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):</p> $\frac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$ <p>Angle formed by three points:</p> $\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$ <p>Line through two points (x_0, y_0) and (x_1, y_1):</p> $\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$ <p>Area of circle, volume of sphere:</p> $A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$		Cartesian	Projective	(x, y)	$(x, y, 1)$	$y = mx + b$	$(m, -1, b)$	$x = c$	$(1, 0, -c)$
Cartesian	Projective								
(x, y)	$(x, y, 1)$								
$y = mx + b$	$(m, -1, b)$								
$x = c$	$(1, 0, -c)$								
<p>If I have seen farther than others, it is because I have stood on the shoulders of giants. – Issac Newton</p>									

Theoretical Computer Science Cheat Sheet

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

Theoretical Computer Science Cheat Sheet

Calculus Cont.

15. $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$
16. $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$
17. $\int \sin^2(ax) dx = \frac{1}{2a} (ax - \sin(ax) \cos(ax)),$
18. $\int \cos^2(ax) dx = \frac{1}{2a} (ax + \sin(ax) \cos(ax)),$
19. $\int \sec^2 x dx = \tan x,$
20. $\int \csc^2 x dx = -\cot x,$
21. $\int \sin^n x dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx,$
22. $\int \cos^n x dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx,$
23. $\int \tan^n x dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x dx, \quad n \neq 1,$
24. $\int \cot^n x dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x dx, \quad n \neq 1,$
25. $\int \sec^n x dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x dx, \quad n \neq 1,$
26. $\int \csc^n x dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x dx, \quad n \neq 1,$
27. $\int \sinh x dx = \cosh x, \quad 28. \int \cosh x dx = \sinh x,$
29. $\int \tanh x dx = \ln |\cosh x|, \quad 30. \int \coth x dx = \ln |\sinh x|, \quad 31. \int \operatorname{sech} x dx = \arctan \sinh x, \quad 32. \int \operatorname{csch} x dx = \ln |\tanh \frac{x}{2}|,$
33. $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2}x, \quad 34. \int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2}x, \quad 35. \int \operatorname{sech}^2 x dx = \tanh x,$
36. $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$
37. $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$
38. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln \left(x + \sqrt{a^2 + x^2} \right), \quad a > 0,$
40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$
41. $\int \sqrt{a^2 - x^2} dx = \frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
42. $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8} (5a^2 - 2x^2) \sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
43. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$
44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|, \quad 45. \int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 - x^2}},$
46. $\int \sqrt{a^2 \pm x^2} dx = \frac{x}{2} \sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left| x + \sqrt{a^2 \pm x^2} \right|,$
47. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left| x + \sqrt{x^2 - a^2} \right|, \quad a > 0,$
48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left| \frac{x}{a+bx} \right|,$
49. $\int x \sqrt{a+bx} dx = \frac{2(3bx - 2a)(a+bx)^{3/2}}{15b^2},$
50. $\int \frac{\sqrt{a+bx}}{x} dx = 2\sqrt{a+bx} + a \int \frac{1}{x\sqrt{a+bx}} dx,$
51. $\int \frac{x}{\sqrt{a+bx}} dx = \frac{1}{\sqrt{2}} \ln \left| \frac{\sqrt{a+bx} - \sqrt{a}}{\sqrt{a+bx} + \sqrt{a}} \right|, \quad a > 0,$
52. $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
53. $\int x \sqrt{a^2 - x^2} dx = -\frac{1}{3} (a^2 - x^2)^{3/2},$
54. $\int x^2 \sqrt{a^2 - x^2} dx = \frac{x}{8} (2x^2 - a^2) \sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
55. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
56. $\int \frac{x dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$
57. $\int \frac{x^2 dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
58. $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left| \frac{a + \sqrt{a^2 + x^2}}{x} \right|,$
59. $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$
60. $\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2},$
61. $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

<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>																																																																																																				

Visual Studio Code

Keyboard shortcuts for Windows

General	Ctrl+M	Toggle Tab moves focus
Search and replace	Ctrl+F	Find
	Ctrl+H	Replace
	F3 / Shift+F3	Find next/previous
	Alt+Shift+F	Select all occurrences of Find match
	Ctrl+P	Quick Open, Go to File...
	Ctrl+Shift+N	New window-instance
	Ctrl+Shift+W	Close window-instance
	Ctrl+,	User Settings
	Ctrl+K Ctrl+S	Keyboard Shortcuts
Basic editing	Ctrl+X	Cut line (empty selection)
	Ctrl+C	Copy line (empty selection)
	Alt+↑ / ↓	Move line up/down
	Shift+Alt+↑ / ↓	Copy line up/down
	Ctrl+Shift+K	Delete line
	Ctrl+Enter	Insert line below
	Ctrl+Shift+Enter	Insert line above
	Ctrl+Shift+\	Jump to matching bracket
	Ctrl+] / [Indent/outdent line
	Home / End	Go to beginning/end of line
	Ctrl+Home	Go to beginning of file
	Ctrl+End	Go to end of file
	Ctrl+↑ / ↓	Scroll line up/down
	Alt+PgUp / PgDn	Scroll page up/down
	Ctrl+Shift+[Fold (collapse) region
	Ctrl+K Ctrl+[Unfold (collapse) region
	Ctrl+K Ctrl+]	Fold (collapse) all subregions
	Ctrl+K Ctrl+]0	Unfold (collapse) all subregions
	Ctrl+K Ctrl+]J	Fold (collapse) all regions
	Ctrl+K Ctrl+C	Add line comment
	Ctrl+K Ctrl+U	Remove line comment
	Ctrl+/	Toggle line comment
	Shift+Alt+A	Toggle block comment
	Alt+Z	Toggle word wrap
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	Go to previous error or warning
	Ctrl+Shift+Tab	Navigate editor group history
	Alt+← / →	Go back / forward
Search and replace	Ctrl+U	Undo last cursor operation
	Shift+Alt+I	Insert cursor above / below
	Ctrl+L	Select current line
	Ctrl+Shift+L	Select all occurrences of current word
	Ctrl+F2	Select all occurrences of current selection
	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
Editor management	Ctrl+`	Show integrated terminal
	Ctrl+Shift+`	Create new terminal
	Ctrl+C	Copy selection
	Ctrl+V	Paste into active terminal
	Ctrl+↑ / ↓	Scroll up/down
	Shift+PgUp / PgDn	Scroll page up/down
	Ctrl+Home / End	Scroll to top/bottom
File management	Ctrl+N	New File
	Ctrl+O	Open File...
	Ctrl+S	Save
	Ctrl+Shift+S	Save As...
	Ctrl+K S	Save All
	Ctrl+E	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
Multi-cursor and selection	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 to the side
	Ctrl+K V	Open Zen Mode (Esc Esc to exit)
	Ctrl+K Z	Zen Mode (Esc Esc to exit)
Debug	F9	Toggle breakpoint
	F5	Start/Continue
	Shift+F5	Stop
	F11 / Shift+F11	Step into/out
	F10	Step over
	Ctrl+K Ctrl+I	Show hover
Integrated terminal	Ctrl+T	Other operating systems' keyboard shortcuts and additional unassigned shortcuts available at aka.ms/vscodekeybindings

Visual Studio Code

Editor management

Multi-cursor and selection

Alt+Click	Insert cursor*
Shift+Alt+↑ / ↓	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+,	Keyboard Shortcuts
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ ↻ / ↻	Move line down/up
Ctrl+Shift+K	Delete line
Ctrl+Enter /	Insert line below/ above
Ctrl+Shift+Enter	
Ctrl+Shift+＼	Jump to matching bracket
Ctrl+] / Ctrl+[Indent/Outdent line
Home / End	Go to beginning/end of line
Ctrl + Home / End	Go to beginning/end of file
Ctrl + 1 / 1	Scroll line up/down
Alt+ PgUp / PgDn	Scroll page up/down
Ctrl+Shift+ [/]	Fold/unfold region
Ctrl+K Ctrl+ [/]	Fold/unfold all subregions
Ctrl+K Ctrl+0 /	Fold/Unfold all regions
Ctrl+K Ctrl+J	Add line comment
Ctrl+K Ctrl+C	Remove line comment
Ctrl+K Ctrl+U	
Ctrl+/	Toggle line comment
Ctrl+Shift+A	Toggle block comment
Alt+Z	Toggle word wrap
Ctrl+K Ctrl+D	
Ctrl+K Ctrl+D	Move last selection to next Find match

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+C	Open new command prompt/terminal
Ctrl+K Ctrl+H	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)

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

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

Integrated terminal

Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+Shift+C	Copy selection
Ctrl+Shift+V	Paste into active terminal
Shift+ PgUp / PgDn	Scroll up/down
Shift+ Home / End	Scroll page up/down
Shift+ Top / Bottom	Scroll to top/bottom

Navigation

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	Peak Definition
Ctrl+K F12	Open Definition to the side
Ctrl+.	Quick Fix
Shift+F12	Go to previous error or warning
F8	Navigate editor group history
Shift+F8	Go to next error or warning
Ctrl+Shift+Tab	Rename Symbol
Ctrl+Alt+-	Trim trailing whitespace
Ctrl+Shift+-	Change file language
Ctrl+M	Toggle Tab moves focus

* The Alt+Click gesture may not work on some Linux distributions.
You can change the modifier key for the "editor.multiCursorModifier" setting.
Ctrl+Click with the "editor.multiCursorModifier" setting.



mitan 2019