



CIRNO  
09

@suzukannn

CIRNO

# All-in at the River

Standard Code Library

Shanghai Jiao Tong University

Desprado2 fstqwq AntiLeaf

# Contents

## 1 数学

1.1 多项式	1
1.1.1 FFT	1
1.1.2 NTT	1
1.1.3 任意模数卷积(MTT, 毛梯梯)	2
1.1.4 多项式操作	3
1.1.5 更优秀的多项式多点求值	5
1.1.6 多项式快速插值	6
1.1.7 拉格朗日反演(多项式复合逆)	7
1.1.8 分治FFT	7
1.1.9 半在线卷积	8
1.1.10 常系数齐次线性递推 $O(k \log k \log n)$	8
1.1.11 应用: $O(\sqrt{n} \log^2 n)$ 快速求阶乘	9
1.2 插值	9
1.2.1 牛顿插值	9
1.2.2 拉格朗日(Lagrange)插值	9
1.3 FWT快速沃尔什变换	9
1.3.1 三行FWT	10
1.4 单纯形	10
1.4.1 线性规划对偶原理	11
1.5 线性代数	11
1.5.1 矩阵乘法	11
1.5.2 高斯消元	11
1.5.3 行列式取模	11
1.5.4 线性基(消成对角)	12
1.5.5 线性代数知识	12
1.5.6 矩阵树定理, BEST定理	12
1.6 博弈论	12
1.6.1 SG定理	12
1.6.2 纳什均衡	12
1.6.3 经典博弈	12
1.6.4 例题	13
1.7 自适应Simpson积分	13
1.8 常见数列	13
1.8.1 斐波那契数 卢卡斯数	13
1.8.2 伯努利数, 自然数幂次和	13
1.8.3 分拆数	13
1.8.4 斯特林数	14
1.8.5 贝尔数	14
1.8.6 欧拉数(Eulerian Number)	15
1.8.7 卡特兰数, 施罗德数, 默慈金数	15
1.9 常用公式及结论	15
1.9.1 方差	15
1.9.2 min-max反演	15
1.9.3 单位根反演(展开整除条件 $[n k]$ )	15
1.9.4 康托展开(排列的排名)	16
1.9.5 连通图计数	16
1.9.6 线性齐次线性常系数递推求通项	16
1.10 常用生成函数变换	16

## 2 数论

2.1 $O(n)$ 预处理逆元	17
2.2 线性筛	17
2.3 杜教筛	17
2.4 Powerful Number筛	17
2.5 洲阁筛	18
2.6 Miller-Rabin	19
2.7 Pollard's Rho	19
2.8 快速阶乘算法	20
2.9 扩展欧几里德	20
2.9.1 求通解的方法	20
2.9.2 类欧几里德算法(直线下整点个数)	20
2.10 中国剩余定理	20

2.10.1 ex-CRT	20
2.11 原根阶	20
2.12 常用数论公式	21
2.12.1 莫比乌斯反演	21
2.12.2 降幂公式	21
2.12.3 其他常用公式	21

## 3 图论

3.1 最小生成树	22
3.1.1 Boruvka算法	22
3.1.2 动态最小生成树	22
3.1.3 最小树形图	23
3.1.4 Steiner Tree 斯坦纳树	24
3.1.5 最小直径生成树	25
3.2 最短路	25
3.2.1 Dijkstra	25
3.2.2 Johnson算法(负权图多源最短路)	25
3.2.3 k短路	25
3.3 Tarjan算法	26
3.3.1 强连通分量	26
3.3.2 割点 点双	26
3.3.3 桥 边双	27
3.4 仙人掌	27
3.4.1 仙人掌DP	27
3.5 二分图	28
3.5.1 匈牙利	28
3.5.2 Hopcroft-Karp二分图匹配	28
3.5.3 KM二分图最大权匹配	29
3.5.4 二分图原理	30
3.6 一般图匹配	30
3.6.1 高斯消元	30
3.6.2 带花树	31
3.6.3 带权带花树	32
3.6.4 原理	34
3.7 支配树	34
3.8 2-SAT	34
3.9 最大流	35
3.9.1 Dinic	35
3.9.2 ISAP	35
3.9.3 HLPP 最高标号预流推进	36
3.10 费用流	37
3.10.1 SPFA费用流	37
3.10.2 Dijkstra费用流	38
3.11 网络流原理	38
3.11.1 最大流	38
3.11.2 最小割	38
3.11.3 费用流	39
3.11.4 上下界网络流	39
3.11.5 常见建图方法	39
3.11.6 例题	39
3.12 Prufer序列	39
3.13 弦图相关	40

## 4 数据结构

4.1 线段树	41
4.1.1 非递归线段树	41
4.1.2 线段树维护矩形并	41
4.2 陈丹琦分治	42
4.2.1 动态图连通性(分治并查集)	42
4.2.2 四维偏序	43
4.3 整体二分	43
4.4 平衡树	44
4.4.1 Treap	44
4.4.2 无旋Treap/可持久化Treap	45
4.4.3 Splay	45

4.5	树链剖分	46	6.1	决策单调性 $O(n \log n)$	73
4.5.1	动态树形DP(最大权独立集)	46	7	杂项	74
4.6	树分治	48	7.1	$O(1)$ 快速乘	74
4.6.1	动态树分治	48	7.2	Kahan求和算法(减少浮点数累加的误差)	74
4.6.2	紫荆花之恋	49	7.3	Python Decimal	74
4.7	LCT动态树	51	7.4	$O(n^2)$ 高精度	74
4.7.1	不换根(弹飞绵羊)	51	7.5	笛卡尔树	77
4.7.2	换根/维护生成树	52	7.6	GarsiaWachs算法( $O(n \log n)$ 合并石子)	77
4.7.3	维护子树信息	53	7.7	常用NTT素数及原根	77
4.7.4	模板题:动态QTREE4(询问树上相距最远点)	55	7.8	xorshift	77
4.8	K-D树	57	7.9	枚举子集	78
4.8.1	动态K-D树(定期重构)	57	7.10	STL	78
4.9	LCA最近公共祖先	58	7.10.1	vector	78
4.9.1	Tarjan LCA $O(n + m)$	58	7.10.2	list	78
4.10	虚树	58	7.10.3	unordered_set/map	78
4.11	长链剖分	59	7.11	Public Based DataStructure(PB_DS)	78
4.11.1	梯子剖分	60	7.11.1	哈希表	78
4.12	左偏树	60	7.11.2	堆	78
4.13	莫队	60	7.11.3	平衡树	78
4.13.1	回滚莫队(无删除莫队)(待完成)	60	7.12	rope	79
4.13.2	莫队二次离线	60	7.13	其他C++相关	79
4.13.3	带修莫队在线化 $O(n^{\frac{5}{3}})$	62	7.13.1	<cmath>	79
4.13.4	莫队二次离线 在线化 $O((n + m)\sqrt{n})$	62	7.13.2	<algorithm>	79
4.14	常见根号思路	63	7.13.3	std::tuple	79
5	字符串	65	7.13.4	<complex>	79
5.1	KMP	65	7.14	一些游戏	79
5.1.1	ex-KMP	65	7.14.1	炉石传说	79
5.2	AC自动机	65	7.15	OEIS	79
5.3	后缀数组	66	7.15.1	计数相关	79
5.3.1	倍增	66	7.15.2	线性递推数列	80
5.3.2	SA-IS	66	7.15.3	数论相关	80
5.3.3	SAMSA	67	7.15.4	其他	80
5.4	后缀平衡树	68	7.16	编译选项	80
5.5	后缀自动机	68	7.17	注意事项	81
5.5.1	广义后缀自动机	68	7.17.1	常见下毒手法	81
5.5.2	区间本质不同子串计数(后缀自动机+LCT+线段树)	69	7.17.2	场外相关	81
5.6	回文树	69	7.17.3	做题策略与心态调节	81
5.6.1	广义回文树	70	7.18	附录: VScode相关	81
5.7	Manacher马拉车	71	7.18.1	插件	81
5.8	字符串原理	72	7.18.2	设置选项	81
6	动态规划	73	7.18.3	快捷键	81
			7.19	附录: 骂人的艺术——梁实秋	81
			7.20	附录: Cheat Sheet	82



# 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)-1.0);
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
10     ↪ 常没有必要用Long double
11
12     Complex(double a = 0.0, double b = 0.0) : a(a), b(b) {}
13
14     Complex operator + (const Complex &x) const {
15         return Complex(a + x.a, b + x.b);
16     }
17
18     Complex operator - (const Complex &x) const {
19         return Complex(a - x.a, b - x.b);
20     }
21
22     Complex operator * (const Complex &x) const {
23         return Complex(a * x.a - b * x.b, a * x.b + b *
24             ↪ x.a);
25     }
26
27     Complex operator * (double x) const {
28         return Complex(a * x, b * x);
29     }
30
31     Complex &operator += (const Complex &x) {
32         return *this = *this + x;
33     }
34
35     Complex conj() const { // 共轭, 一般只有NTT需要用
36         return Complex(a, -b);
37     }
38 } omega[maxn], omega_inv[maxn];
39 const Complex ima = Complex(0, 1);
40
41 int fft_n; // 要在主函数里初始化
42
43 // FFT初始化
44 void FFT_init(int n) {
45     fft_n = n;
46
47     for (int i = 0; i < n; i++) // 根据单位根的旋转性质可以
48         ↪ 节省计算单位根逆元的时间
49         omega[i] = Complex(cos(2 * pi / n * i), sin(2 * pi
50             ↪ / n * i));
51
52     omega_inv[0] = omega[0];
53     for (int i = 1; i < n; i++)
54         omega_inv[i] = omega[n - i];
55     // 当然不存单位根也可以, 只不过在FFT次数较多时很可能会
56     ↪ 增大常数
57 }
58
59 // FFT主过程
60 void FFT(Complex *a, int n, int tp) {
61     for (int i = 1, j = 0, k; i < n - 1; i++) {
62         k = n;
63         do
64             j ^= (k >= 1);
65             while (j < k);
66
67         if (i < j)
68             swap(a[i], a[j]);
69     }
70
71     for (int k = 2, m = fft_n / 2; k <= n; k *= 2, m /= 2)
72         for (int i = 0; i < n; i += k)
73             for (int j = 0; j < k / 2; j++) {
74                 Complex u = a[i + j], v = (tp > 0 ? omega :
75                     ↪ omega_inv)[m * j] * a[i + j + k / 2];
76
77                 a[i + j] = u + v;
78                 a[i + j + k / 2] = u - v;
79             }
80
81     if (tp < 0)
82         for (int i = 0; i < n; i++) {
83             a[i].a /= n;
84             a[i].b /= n; // 一般情况下是不需要的, 只有NTT时
85             ↪ 才需要
86         }
87 }

```

```

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 :
70                 ↪ omega_inv)[m * j] * a[i + j + k / 2];
71
72             a[i + j] = u + v;
73             a[i + j + k / 2] = u - v;
74         }
75
76 if (tp < 0)
77     for (int i = 0; i < n; i++) {
78         a[i].a /= n;
79         a[i].b /= n; // 一般情况下是不需要的, 只有NTT时
80         ↪ 才需要
81     }
82 }

```

### 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
4     ↪ 里初始化
5
6 void NTT_init(int n) {
7     ntt_n = n;
8
9     int wn = qpow(3, (p - 1) / n); // 这里的3代表模数的任意
10     ↪ 一个原根
11
12     omega[0] = omega_inv[0] = 1;
13
14     for (int i = 1; i < n; i++)
15         omega_inv[n - i] = omega[i] = (long long)omega[i -
16             ↪ 1] * wn % p;
17 }
18
19 void NTT(int *a, int n, int tp) { // n为变换长度,
20     ↪ tp为1或-1, 表示正/逆变换
21
22     for (int i = 1, j = 0, k; i < n - 1; i++) { //  $O(n)$ 旋转
23         ↪ 算法, 原理是模拟加1
24         k = n;
25         do
26             j ^= (k >= 1);
27             while (j < k);
28
29         if (i < j)
30             swap(a[i], a[j]);
31     }
32
33     for (int k = 2, m = ntt_n / 2; k <= n; k *= 2, m /= 2)
34         for (int i = 0; i < n; i += k)
35             for (int j = 0; j < k / 2; j++) {
36                 int w = (tp > 0 ? omega : omega_inv)[m *
37                     ↪ j];
38
39                 int u = a[i + j], v = (long long)w * a[i +
40                     ↪ j + k / 2] % p;
41                 a[i + j] = u + v;
42                 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     }
48 }

```

### 1.1.3 任意模数卷积(MTT, 毛梯梯)

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

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

只需要DFT一个, 另一个DFT实际上就是前者反转再取共轭, 再利用

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

即可还原出 $A(x)$ ,  $B(x)$ .

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

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

```

1 // 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
25 void IDFT(Complex *a, Complex *b, int n) {
26     static Complex c[maxn];
27
28     for (int i = 0; i < n; i++)
29         c[i] = a[i] + ima * b[i];
30
31     FFT(c, n, -1);
32
33     for (int i = 0; i < n; i++) {
34         a[i].a = c[i].a;

```

```

35         b[i].a = c[i].b;
36     }
37 }
38
39 Complex a[2][maxn], b[2][maxn], c[3][maxn];
40 int ans[maxn];
41
42 int main() {
43     int n, m;
44     scanf("%d%d", &n, &m, &p);
45     n++;
46     m++;
47
48     base = (int)(sqrt(p) + 0.5);
49
50     for (int i = 0; i < n; i++) {
51         int x;
52         scanf("%d", &x);
53         x %= p;
54
55         a[1][i].a = x / base;
56         a[0][i].a = x % base;
57     }
58
59     for (int i = 0; i < m; i++) {
60         int x;
61         scanf("%d", &x);
62         x %= p;
63
64         b[1][i].a = x / base;
65         b[0][i].a = x % base;
66     }
67
68     int N = 1;
69     while (N < n + m - 1)
70         N <<= 1;
71
72     FFT_init(N);
73
74     DFT(a[0], a[1], N);
75     DFT(b[0], b[1], N);
76
77     for (int i = 0; i < N; i++)
78         c[0][i] = a[0][i] * b[0][i];
79
80     for (int i = 0; i < N; i++)
81         c[1][i] = a[0][i] * b[1][i] + a[1][i] * b[0][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], 这样
95     ↳ 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             printf(" ");
100         printf("%d", ans[i]);

```

```

101     return 0;
102 }

```

### 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
8     ↪ 为1就可以
9
10    for (int k = 2; k <= n; k <= 1) {
11        memcpy(B, A, sizeof(int) * k);
12        memset(B + k, 0, sizeof(int) * k);
13
14        NTT(B, k * 2, 1);
15        NTT(C, k * 2, 1);
16
17        for (int i = 0; i < k * 2; i++) {
18            C[i] = (2 - (long long)B[i] * C[i]) % p * C[i]
19            ↪ % p;
20            if (C[i] < 0)
21                C[i] += p;
22        }
23
24        NTT(C, k * 2, -1);
25
26        memset(C + k, 0, sizeof(int) * k);
27    }
28 // 开根
29 void get_sqrt(int *A, int *C, int n) {
30     static int B[maxn], D[maxn];
31
32     memset(C, 0, sizeof(int) * (n * 2));
33     C[0] = 1; // 如果不是1就要考虑二次剩余
34
35     for (int k = 2; k <= n; k *= 2) {
36         memcpy(B, A, sizeof(int) * k);
37         memset(B + k, 0, sizeof(int) * k);
38
39         get_inv(C, D, k);
40
41         NTT(B, k * 2, 1);
42         NTT(D, k * 2, 1);
43
44         for (int i = 0; i < k * 2; i++)
45             B[i] = (long long)B[i] * D[i] % p;
46
47         NTT(B, k * 2, -1);
48
49         for (int i = 0; i < k; i++)
50             C[i] = (long long)(C[i] + B[i]) * inv_2 % p; //
51             ↪ inv_2是2的逆元
52     }
53
54 // 求导
55 void get_derivative(int *A, int *C, int n) {
56     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
72     ↪ 是1
73     static int B[maxn];
74
75     get_derivative(A, B, n);
76     memset(B + n, 0, sizeof(int) * n);
77
78     get_inv(A, C, n);
79
80     NTT(B, n * 2, 1);
81     NTT(C, n * 2, 1);
82
83     for (int i = 0; i < n * 2; i++)
84         B[i] = (long long)B[i] * C[i] % p;
85
86     NTT(B, n * 2, -1);
87
88     get_integrate(B, C, n);
89
90     memset(C + n, 0, sizeof(int) * n);
91 }
92
93 // 多项式exp, 要求A没有常数项
94 // 常数很大且总代码较长, 一般来说最好替换为分治FFT
95 // 分治FFT依据: 设 $G(x) = \exp F(x)$ , 则有  $g_i = \sum_{k=1}^i f_{i-k} g_k$ 
96 ↪  $\wedge^{i-1} f_{i-k} g_k$ 
97 void get_exp(int *A, int *C, int n) {
98     static int B[maxn];
99
100     memset(C, 0, sizeof(int) * (n * 2));
101     C[0] = 1;
102
103     for (int k = 2; k <= n; k <= 1) {
104         get_ln(C, B, k);
105
106         for (int i = 0; i < k; i++) {
107             B[i] = A[i] - B[i];
108             if (B[i] < 0)
109                 B[i] += p;
110         }
111         (++B[0]) %= p;
112
113         NTT(B, k * 2, 1);
114         NTT(C, k * 2, 1);
115
116         for (int i = 0; i < k * 2; i++)
117             C[i] = (long long)C[i] * B[i] % p;
118
119         NTT(C, k * 2, -1);
120
121         memset(C + k, 0, sizeof(int) * k);
122     }
123 }
124
125 // 多项式k次幂, 在A常数项不为1时需要转化
126 // 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
127 ↪ 速幂
128 void get_pow(int *A, int *C, int n, int k) {
129     static int B[maxn];
130
131     get_ln(A, B, n);
132

```

```

129     for (int i = 0; i < n; i++)
130     |   B[i] = (long long)B[i] * k % p;
131
132
133     get_exp(B, C, n);
134 }
135
136 // 多项式除法,  $A / B$ , 结果输出在C
137 // A的次数为 $n$ , B的次数为 $m$ 
138 void get_div(int *A, int *B, int *C, int n, int m) {
139     static int f[maxn], g[maxn], gi[maxn];
140
141     if (n < m) {
142         memset(C, 0, sizeof(int) * m);
143         return;
144     }
145
146     int N = 1;
147     while (N < (n - m + 1))
148         N <<= 1;
149
150     memset(f, 0, sizeof(int) * N * 2);
151     memset(g, 0, sizeof(int) * N * 2);
152     // memset(gi, 0, sizeof(int) * N);
153
154     for (int i = 0; i < n - m + 1; i++)
155         f[i] = A[n - i - 1];
156     for (int i = 0; i < m && i < n - m + 1; i++)
157         g[i] = B[m - i - 1];
158
159     get_inv(g, gi, N);
160
161     for (int i = n - m + 1; i < N; i++)
162         gi[i] = 0;
163
164     NTT(f, N * 2, 1);
165     NTT(gi, N * 2, 1);
166
167     for (int i = 0; i < N * 2; i++)
168         f[i] = (long long)f[i] * gi[i] % p;
169
170     NTT(f, N * 2, -1);
171
172     for (int i = 0; i < n - m + 1; i++)
173         C[i] = f[n - m - i];
174 }
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 m)
178     ↪ {
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     if (D) { // D是商, 可以选择不要
193         for (int i = 0; i < n - m + 1; i++)
194             D[i] = d[i];
195     }
196
197     int N = 1;
198     while (N < n)
199         N *= 2;
200
201     memcpy(b, B, sizeof(int) * m);
202
203     NTT(b, N, 1);
204     NTT(d, N, 1);
205
206     for (int i = 0; i < N; i++)
207         b[i] = (long long)d[i] * b[i] % p;
208
209     NTT(b, N, -1);
210
211     for (int i = 0; i < m - 1; i++)
212         C[i] = (A[i] - b[i] + p) % p;
213
214     memset(b, 0, sizeof(int) * N);
215     memset(d, 0, sizeof(int) * N);
216 }
217
218 // 多点求值要用的数组
219 int q[maxn], ans[maxn]; // q是要代入的各个系数, ans是求出的
220     ↪ 值
221 int tg[25][maxn * 2], tf[25][maxn]; // 辅助数组, tg是预处理
222     ↪ 乘积,
223 // tf是项数越来越少的f, tf[0]就是原来的函数
224
225 void pretreat(int l, int r, int k) { // 多点求值预处理
226     static int A[maxn], B[maxn];
227
228     int *g = tg[k] + l * 2;
229
230     if (r - l + 1 <= 200) {
231         g[0] = 1;
232
233         for (int i = l; i <= r; i++) {
234             for (int j = i - l + 1; j; j--) {
235                 g[j] = (g[j - 1] - (long long)g[j] * q[i])
236                     ↪ % p;
237                 if (g[j] < 0)
238                     g[j] += p;
239             }
240             g[0] = (long long)g[0] * (p - q[i]) % p;
241         }
242
243         return;
244     }
245
246     int mid = (l + r) / 2;
247
248     pretreat(l, mid, k + 1);
249     pretreat(mid + 1, r, k + 1);
250
251     if (!k)
252         return;
253
254     int N = 1;
255     while (N <= r - l + 1)
256         N *= 2;
257
258     int *gl = tg[k + 1] + l * 2, *gr = tg[k + 1] + (mid +
259     ↪ 1) * 2;
260
261     memset(A, 0, sizeof(int) * N);
262     memset(B, 0, sizeof(int) * N);
263
264     memcpy(A, gl, sizeof(int) * (mid - l + 2));
265     memcpy(B, gr, sizeof(int) * (r - mid + 1));
266
267     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 - 1 + 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 - 1; ~j; j--)
282                 ans[i] = ((long long)ans[i] * x + f[j]) %
283                     ↪ p;
284
285             return;
286         }
287
288         int mid = (l + r) / 2;
289         int *ff = tf[k + 1], *gl = tg[k + 1] + 1 * 2, *gr =
290             ↪ tg[k + 1] + (mid + 1) * 2;
291
292         get_mod(f, gl, ff, NULL, 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, NULL, 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
305     // f < x^n, m个询问, 询问是0-based, 当然改成1-based也很简单
306     void get_value(int *f, int *x, int *a, int n, int m) {
307         if (m <= n)
308             m = n + 1;
309         if (n < m - 1)
310             n = m - 1; // 补零方便处理
311
312         memcpy(tf[0], f, sizeof(int) * n);
313         memcpy(q, x, sizeof(int) * m);
314
315         pretreat(0, m - 1, 0);
316         solve(0, m - 1, 0);
317
318         if (a) // 如果a是NULL, 代表不复制答案, 直接用ans数组
319             memcpy(a, ans, sizeof(int) * m);
320     }

```

### 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] + 1 * 2;
9
10    if (r - l + 1 <= 1) {
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])
16                    ↪ % p;
17                if (g[j] < 0)
18                    g[j] += p;
19            }
20            g[0] = (long long)g[0] * (p - q[i]) % p;
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] + 1 * 2, *gr = tg[k + 1] + (mid +
37        ↪ 1) * 2;
38
39    memset(A, 0, sizeof(int) * N);
40    memset(B, 0, sizeof(int) * N);
41
42    memcpy(A, gl, sizeof(int) * (mid - l + 2));
43    memcpy(B, gr, sizeof(int) * (r - mid + 1));
44
45    NTT(A, N, 1);
46    NTT(B, N, 1);
47
48    for (int i = 0; i < N; i++)
49        A[i] = (long long)A[i] * B[i] % p;
50
51    NTT(A, N, -1);
52
53    for (int i = 0; i <= r - l + 1; i++)
54        g[i] = A[i];
55 }
56
57 void solve(int l, int r, int k) { // 主过程
58     static int a[maxn], b[maxn];
59
60     int *f = tf[k];
61
62     if (l == r) {
63         ans[l] = f[0];
64         return;
65     }
66
67     int mid = (l + r) / 2;
68     int *ff = tf[k + 1], *gl = tg[k + 1] + 1 * 2, *gr =
69         ↪ tg[k + 1] + (mid + 1) * 2;

```



```

69     int N = 1;
70     while (N < r - l + 2)
71         N *= 2;
72
73     memcpy(a, f, sizeof(int) * (r - l + 2));
74     memcpy(b, gr, sizeof(int) * (r - mid + 1));
75     reverse(b, b + r - mid + 1);
76
77     NTT(a, N, 1);
78     NTT(b, N, 1);
79     for (int i = 0; i < N; i++)
80         b[i] = (long long)a[i] * b[i] % p;
81
82     reverse(b + 1, b + N);
83     NTT(b, N, 1);
84     int n_inv = qpow(N, p - 2);
85     for (int i = 0; i < N; i++)
86         b[i] = (long long)b[i] * n_inv % p;
87
88     for (int i = 0; i < mid - l + 2; i++)
89         ff[i] = b[i + r - mid];
90
91     memset(a, 0, sizeof(int) * N);
92     memset(b, 0, sizeof(int) * N);
93
94     solve(l, mid, k + 1);
95
96     memset(ff, 0, sizeof(int) * (mid - l + 2));
97
98     memcpy(a, f, sizeof(int) * (r - l + 2));
99     memcpy(b, gl, sizeof(int) * (mid - l + 2));
100    reverse(b, b + mid - l + 2);
101
102    NTT(a, N, 1);
103    NTT(b, N, 1);
104    for (int i = 0; i < N; i++)
105        b[i] = (long long)a[i] * b[i] % p;
106
107    reverse(b + 1, b + N);
108    NTT(b, N, 1);
109    for (int i = 0; i < N; i++)
110        b[i] = (long long)b[i] * n_inv % p;
111
112    for (int i = 0; i < r - mid + 1; i++)
113        ff[i] = b[i + mid - l + 1];
114
115    memset(a, 0, sizeof(int) * N);
116    memset(b, 0, sizeof(int) * N);
117
118    solve(mid + 1, r, k + 1);
119
120    memset(gl, 0, sizeof(int) * (mid - l + 2));
121    memset(gr, 0, sizeof(int) * (r - mid + 1));
122    memset(ff, 0, sizeof(int) * (r - mid + 1));
123 }
124
125 // f < x^n, m个询问, theta-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
136     pretreat(0, m - 1, 0);
137

```

```

138     int N = 1;
139     while (N < m)
140         N *= 2;
141
142     get_inv(tg[0], c, N);
143
144     fill(c + m, c + N, 0);
145     reverse(c, c + m);
146
147     memcpy(d, f, sizeof(int) * m);
148
149     NTT(c, N * 2, 1);
150     NTT(d, N * 2, 1);
151     for (int i = 0; i < N * 2; i++)
152         c[i] = (long long)c[i] * d[i] % p;
153     NTT(c, N * 2, -1);
154
155     for (int i = 0; i < m; i++)
156         tf[0][i] = c[i + n];
157
158     solve(0, m - 1, 0);
159
160     if (a) // 如果a是NULL, 代表不复制答案, 直接用ans数组
161         memcpy(a, ans, sizeof(int) * m);
162 }

```

### 1.1.6 多项式快速插值

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

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

第一步要先对每个 $i$ 求出

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

设

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

那么想要的是

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

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

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

设

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

第二步要求出

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

使用分治. 设

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

则原式化为

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

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

递归计算, 复杂度  $O(n \log^2 n)$ .

注意由于整体和局部的  $M(x)$  都要用到, 要预处理一下.

```

1 int qx[maxn], qy[maxn];
2 int th[25][maxn * 2], ansf[maxn]; // th存的是各阶段的M(x)
3
4 void pretreat2(int l, int r, int k) { // 预处理
5     static int A[maxn], B[maxn];
6     int *h = th[k] + 1 * 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] + 1 * 2, *hr = th[k + 1] + (mid +
24        ↪ 1) * 2;
25
26    memset(A, 0, sizeof(int) * N);
27    memset(B, 0, sizeof(int) * N);
28
29    memcpy(A, hl, sizeof(int) * (mid - l + 2));
30    memcpy(B, hr, sizeof(int) * (r - mid + 1));
31
32    NTT(A, N, 1);
33    NTT(B, N, 1);
34
35    for (int i = 0; i < N; i++)
36        A[i] = (long long)A[i] * B[i] % p;
37
38    NTT(A, N, -1);
39
40    for (int i = 0; i <= r - l + 1; i++)
41        h[i] = A[i];
42
43    void solve2(int l, int r, int k) { // 分治
44        static int A[maxn], B[maxn], t[maxn];
45
46        if (l == r)
47            return;
48
49        int mid = (l + r) / 2;
50
51        solve2(l, mid, k + 1);
52        solve2(mid + 1, r, k + 1);
53
54        int *hl = th[k + 1] + 1 * 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);

```

```

62    memset(B, 0, sizeof(int) * N);
63
64    memcpy(A, ansf + 1, sizeof(int) * (mid - l + 1));
65    memcpy(B, hr, sizeof(int) * (r - mid + 1));
66
67    NTT(A, N, 1);
68    NTT(B, N, 1);
69
70    for (int i = 0; i < N; i++)
71        t[i] = (long long)A[i] * B[i] % p;
72
73    memset(A, 0, sizeof(int) * N);
74    memset(B, 0, sizeof(int) * N);
75
76    memcpy(A, ansf + mid + 1, sizeof(int) * (r - mid));
77    memcpy(B, hl, sizeof(int) * (mid - l + 2));
78
79    NTT(A, N, 1);
80    NTT(B, N, 1);
81
82    for (int i = 0; i < N; i++)
83        t[i] = (t[i] + (long long)A[i] * B[i]) % p;
84
85    NTT(t, N, -1);
86
87    memcpy(ansf + 1, t, sizeof(int) * (r - l + 1));
88 }
89
90 // 主过程
91 // 如果x, y传NULL表示询问已经存在了qx, qy里
92 void interpolation(int *x, int *y, int n, int *f = NULL) {
93     static int d[maxn];
94
95     if (x)
96         memcpy(qx, x, sizeof(int) * n);
97     if (y)
98         memcpy(qy, y, sizeof(int) * n);
99
100    pretreat2(0, n - 1, 0);
101
102    get_derivative(th[0], d, n + 1);
103
104    multipoint_eval(d, qx, NULL, n, n);
105
106    for (int i = 0; i < n; i++)
107        ansf[i] = (long long)qy[i] * qpow(ans[i], p - 2) %
108            ↪ p;
109
110    solve2(0, n - 1, 0);
111
112    if (f)
113        memcpy(f, ansf, sizeof(int) * n);
114 }

```

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

```

```

39     B[i - l] = (c[i] + g[i]) % p;
40     NTT(A, r - l, 1);
41     NTT(B, r - l, 1);
42     for (int i = 0; i < r - l; i++)
43         A[i] = (long long)A[i] * B[i] % p;
44     NTT(A, r - l, -1);
45
46     for (int i = mid; i < r; i++)
47         f[i] = (f[i] + A[i - l]) % p;
48
49     memset(A, 0, sizeof(int) * (r - l));
50     memset(B, 0, sizeof(int) * (r - l));
51
52     for (int i = l; i < mid; i++)
53         A[i - l] = f[i];
54     for (int i = 0; i < r - l; i++)
55         B[i] = (c[i] + g[i]) % p;
56     NTT(A, r - l, 1);
57     NTT(B, r - l, 1);
58     for (int i = 0; i < r - l; i++)
59         A[i] = (long long)A[i] * B[i] % p;
60     NTT(A, r - l, -1);
61
62     for (int i = mid; i < r; i++)
63         f[i] = (f[i] + A[i - l]) % p;
64 }
65
66 memset(A, 0, sizeof(int) * (r - l));
67 memset(B, 0, sizeof(int) * (r - l));
68
69 solve(mid, r);
70 }

```

### 1.1.10 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 m)
3     ↪ {
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    }
15
16    get_div(A, B, d, n, m);
17
18    if (D) { // D是商, 可以选择不要
19        for (int i = 0; i < n - m + 1; i++)
20            D[i] = d[i];
21    }
22
23    int N = 1;
24    while (N < n)
25        N *= 2;
26
27    if (!flag) {
28        memcpy(b, B, sizeof(int) * m);
29        NTT(b, N, 1);
30
31        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 }
46
47 // g < x^n, f是输出答案的数组
48 void pow_mod(long long k, int *g, int n, int *f) {
49     static int a[maxn], t[maxn];
50
51     memset(f, 0, sizeof(int) * (n * 2));
52
53     f[0] = a[1] = 1;
54
55     int N = 1;
56     while (N < n * 2 - 1)
57         N *= 2;
58
59     while (k) {
60         NTT(a, N, 1);
61
62         if (k & 1) {
63             memcpy(t, f, sizeof(int) * N);
64
65             NTT(t, N, 1);
66             for (int i = 0; i < N; i++)
67                 t[i] = (long long)t[i] * a[i] % p;
68             NTT(t, N, -1);
69
70             get_mod(t, g, f, NULL, n * 2 - 1, n);
71         }
72
73         for (int i = 0; i < N; i++)
74             a[i] = (long long)a[i] * a[i] % p;
75         NTT(a, N, -1);
76
77         memcpy(t, a, sizeof(int) * (n * 2 - 1));
78         get_mod(t, g, a, NULL, n * 2 - 1, n);
79         fill(a + n - 1, a + N, 0);
80
81         k >>= 1;
82     }
83
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 *a) {
90     static int g[maxn], c[maxn];
91
92     memset(g, 0, sizeof(int) * (m * 2 + 1));
93
94     for (int i = 0; i < m; i++)
95         g[i] = (p - a[m - i]) % p;
96     g[m] = 1;
97
98     pow_mod(n, g, m + 1, c);
99
100     int ans = 0;

```

```

101     for (int i = 0; i < m; i++)
102         ans = (ans + (long long)c[i] * f[i]) % p;
103
104     return ans;
105 }

```

### 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

```

1 // 注意FWT常数比较小, 这点与FFT/NTT不同
2 // 以下代码均以模质数情况为例, 其中n为变换长度, tp表示
  // 正/逆变换
3
4 // 按位或版本
5 void FWT_or(int *A, int n, int tp) {

```

```

6   for (int k = 2; k <= n; k *= 2)
7       for (int i = 0; i < n; i += k)
8           for (int j = 0; j < k / 2; j++) {
9               if (tp > 0)
10                  A[i + j + k / 2] = (A[i + j + k / 2] +
11                                     ↪ A[i + j]) % p;
12               else
13                  A[i + j + k / 2] = (A[i + j + k / 2] -
14                                     ↪ A[i + j] + p) % p;
15           }
16 }
17 // 按位与版本
18 void FWT_and(int *A, int n, int tp) {
19     for (int k = 2; k <= n; k *= 2)
20         for (int i = 0; i < n; i += k)
21             for (int j = 0; j < k / 2; j++) {
22                 if (tp > 0)
23                     A[i + j] = (A[i + j] + A[i + j + k /
24                               ↪ 2]) % p;
25                 else
26                     A[i + j] = (A[i + j] - A[i + j + k / 2]
27                               ↪ + p) % p;
28             }
29 }
30 // 按位异或版本
31 void FWT_xor(int *A, int n, int tp) {
32     for (int k = 2; k <= n; k *= 2)
33         for (int i = 0; i < n; i += k)
34             for (int j = 0; j < k / 2; j++) {
35                 int a = A[i + j], b = A[i + j + k / 2];
36                 A[i + j] = (a + b) % p;
37                 A[i + j + k / 2] = (a - b + p) % p;
38             }
39     if (tp < 0) {
40         int inv = qpow(n % p, p - 2); // n的逆元, 在不取模
41         ↪ 时需要用每层除以2代替
42         for (int i = 0; i < n; i++)
43             A[i] = A[i] * inv % p;
44     }
45 }

```

### 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

```

```
24 // xor同理
```

## 1.4 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 (!inititalize())
23        printf("Infeasible"); // 无解
24    else if (!simplex())
25        printf("Unbounded"); // 最优解无限大
26
27    else {
28        printf("%.15lf\n", -A[0][0]);
29        if (t) {
30            for (int i = 1; i <= m; i++)
31                x[id[i + n]] = A[i][0];
32            for (int i = 1; i <= n; i++)
33                printf("%.15lf ", x[i]);
34        }
35    }
36    return 0;
37 }
38
39 //初始化
40 //对于初始解可行的问题,可以把初始化省略掉
41 bool inititalize() {
42     while (true) {
43         double t = 0.0;
44         int l = 0, e = 0;
45
46         for (int i = 1; i <= m; i++)
47             if (A[i][0] + eps < t) {
48                 t = A[i][0];
49                 l = i;
50             }
51
52         if (!l)
53             return true;
54
55         for (int i = 1; i <= n; i++)
56             if (A[l][i] < -eps && (!e || id[i] < id[e]))
57                 e = i;
58
59         if (!e)
60             return false;
61
62         pivot(l, e);
63     }
64 }

```



```

65 //求解
66 bool simplex() {
67     while (true) {
68         int l = 0, e = 0;
69         for (int i = 1; i <= n; i++)
70             if (A[0][i] > eps && (!e || id[i] < id[e]))
71                 e = i;
72
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                 l = i;
81                 t = A[i][0]/A[i][e];
82             }
83
84         if (!l)
85             return false;
86
87         pivot(l, e);
88     }
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 + 1]);
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 }

```

1.4.1 线性规划对偶原理

给定一个原始线性规划:

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

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:

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

1.5 线性代数

1.5.1 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

```

1.5.2 高斯消元

高斯-约当消元法 Gauss-Jordan

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

```

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,如果有可能无解
9             ↪ 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][k];
17    }
18 }

```

解线性方程组

在矩阵的右边加上一列表示系数即可,如果消成上三角的话最后要倒序回代.

求逆矩阵

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

行列式

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

1.5.3 行列式取模

```

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 long)A[j][k]
12                        ↪ * t) % p;
13
14                swap(A[i], A[j]);
15            }
16    }
17 }

```

```
14         det = (p - det) % p; // 交换一次之后行列式
           ↪ 取负
15     }
16
17     if (!A[i][i])
18         return 0;
19
20     det = (long long)det * A[i][i] % p;
21
22 }
23
24 return det;
25 }
```

#### 1.5.4 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 }
```

#### 1.5.5 线性代数知识

行列式:

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

逆矩阵:

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

代数余子式:

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

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).

#### 1.5.6 矩阵树定理, BEST定理

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

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

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

(可以看出, 只有无向图才满足 $L(G)$ 的所有代数余子式都相等.)

**BEST定理(有向图欧拉回路计数):** 如果 $G$ 是有向欧拉图, 则 $G$ 的欧拉回路的个数等于以任意一个点为根的内/外向树个数乘以 $\prod_v (\deg(v) - 1)!$ .

并且在欧拉图里, 无论以哪个结点为根, 也无论内向树还是外向树, 个数都是一样的.

另外无向图欧拉回路计数是NP问题.

## 1.6 博弈论

### 1.6.1 SG定理

对于一个平等游戏, 可以为每个状态定义一个SG函数.

一个状态的SG函数等于所有它能一步到达的状态的SG函数的mex, 也就是最小的没有出现过的自然数.

那么所有先手必败态的SG函数为0, 先手必胜态的SG函数非0.

如果有一个游戏, 它由若干个独立的子游戏组成, 且每次行动时只能选一个子游戏进行操作, 则这个游戏的SG函数就是所有子游戏的SG函数的异或和. (比如最经典的Nim游戏, 每次只能选一堆取若干个石子.)

同时操作多个子游戏的结论参见“经典博弈”部分.

### 1.6.2 纳什均衡

首先定义纯策略和混合策略: 纯策略是指你一定会选择某个选项, 混合策略是指你对每个选项都有一个概率分布 $p_i$ , 你会以相应的概率选择这个选项.

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

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

纳什均衡点可能存在多个, 不过在一个双人零和游戏中, 纳什均衡点一定唯一存在.

### 1.6.3 经典博弈

#### 1. 阶梯博弈

台阶的每层都有一些石子, 每次可以选一层(但不能是第0层), 把任意个石子移到低一层.

**结论:** 奇数层的石子数量进行异或和即可.

实际上只要路径长度唯一就可以, 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, 唯一的区别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：**

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

### 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。

#### 1.6.4 例题

##### 1. 黑白棋游戏

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

**结论：**只需要看最左边的棋子即可，因为每次操作最左边的棋子一定会被翻转。

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

## 1.7 自适应Simpson积分

Forked from fstqwq's template.

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

## 1.8 常见数列

查表参见“Miscellaneous/OEIS”部分。

### 1.8.1 斐波那契数 卢卡斯数

斐波那契数:  $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

卢卡斯数:  $L_0 = 2, L_1 = 1$

2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, ...

### 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) return {0, 1};
3     auto p = fib(n >> 1);
4     int c = p.first * (2 * p.second - p.first);
5     int d = p.first * p.first + p.second * p.second;
6     if (n & 1)
7         return {d, c + d};
8     else
9         return {c, d};
10 }
```

### 1.8.2 伯努利数，自然数幂次和

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

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

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

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

$B_0 = 1, B_1 = -\frac{1}{2}, B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, \dots$   
(除了 $B_1 = -\frac{1}{2}$ 以外，伯努利数的奇数项都是0.)

自然数幂次和关于次数的EGF:

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

### 1.8.3 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;
24     i++) /*递推式系数1, 2, 5, 7, 12, 15, 22, 26... i*(3*i-1)/
           ↪ 2, i*(3*i+1)/2*/
25 {
26     a[2 * i] = i * (i * 3 - 1) / 2; /*五边形数
           ↪ 为1, 5, 12, 22... i*(3*i-1)/2*/
27     a[2 * i + 1] = i * (i * 3 + 1) / 2;
28 }
29 for (
30     i = 3; i < 50005;
31     i++) /*p[n]=p[n-1]+p[n-2]-p[n-5]-
           ↪ p[n-7]+p[12]+p[15]-...+p[n-i*[3i-1]/2]+p[n-
           ↪ i*[3i+1]/2]*/
32 {
33     p[i] = 0;
34     int j;
35     for (j = 2; a[j] <= i; j++) /*有可能为负数, 式中
           ↪ 加1000007*/
36     {
37         if (j & 2) {
38             p[i] = (p[i] + p[i - a[j]] + 1000007) % 1000007;
39         } else {
40             p[i] = (p[i] - p[i - a[j]] + 1000007) % 1000007;
41         }
42     }
43 }
44 int n;
45 while (~scanf("%d", &n))
46     printf("%lld\n", p[n]);
47 }

```

### 3. 斯特林反演

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

### 4. 幂的转换

#### 上升幂与普通幂的转换

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

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

#### 下降幂与普通幂的转换

$$x^n = \sum_k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\underline{k}} = \sum_k \binom{x}{k} \left\{ \begin{matrix} n \\ k \end{matrix} \right\} k!$$

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

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:

$$\sigma_n(x) = \frac{\left[ \begin{matrix} x \\ n \end{matrix} \right]}{x(x-1)\dots(x-n)}$$

$\sigma_n(x)$ 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.

$$\left[ \begin{matrix} n \\ n-k \end{matrix} \right] = n^{\overline{k+1}} \sigma_k(n)$$

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

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.

### 1.8.5 贝尔数

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

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

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

递推式:

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

指数生成函数:

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

Touchard同余:

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

### 1.8.4 斯特林数

#### 1. 第一类斯特林数

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

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

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

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

求同一列: 用一个轮换的指数生成函数做 $k$ 次幂

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

#### 2. 第二类斯特林数

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

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

求一个: 容斥, 狗都会做

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

求同一行: FFT, 狗都会做

求同一列: 指数生成函数

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

普通生成函数

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

1.8.6 欧拉数(Eulerian Number)

1. 欧拉数

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

$$\langle n \rangle_k = (n - k) \langle n - 1 \rangle_{k - 1} + (k + 1) \langle n - 1 \rangle_k$$

$$\langle n \rangle_k = \sum_{i=0}^{k+1} (-1)^i \binom{n+1}{i} (k+1-i)^n$$

$$\sum_{k=0}^{n-1} \langle n \rangle_k = n!$$

$$x^n = \sum_{k=0}^{n-1} \langle n \rangle_k \binom{x+k}{n}$$

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

2. 二阶欧拉数

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

$$\langle\langle n \rangle\rangle_k = (2n - k - 1) \langle\langle n - 1 \rangle\rangle_{k - 1} + (k + 1) \langle\langle n - 1 \rangle\rangle_k$$

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

3. 二阶欧拉数与斯特林数的关系

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

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

1.8.7 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.

1.9 常用公式及结论

1.9.1 方差

$m$ 个数的方差:

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

随机变量的方差:  $D^2(x) = E(x^2) - E^2(x)$

1.9.2 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.

1.9.3 单位根反演(展开整除条件 $[n|k]$ )

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

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



1.9.4 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.

1.9.5 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.

1.9.6 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). 
$$U(x) = \sum_{n \geq 0} u_n x^n = \frac{h(x)}{1 - \sum_{j=1}^r c_j x^j}, \quad \deg(h(x)) < r.$$

(ii). 若特征多项式

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

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

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

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

1.10 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

```

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

```

### 2.2 线性筛

```

1 // 此代码以计算约数之和函数\sigma_1(对10^9+7取模)为例
2 // 适用于任何f(p^k)便于计算的积性函数
3 constexpr int p = 1000000007;
4
5 int prime[maxn / 10], sigma_one[maxn], f[maxn], g[maxn];
6 // f: 除掉最小质因子后剩下的部分
7 // g: 最小质因子的幂次, 在f(p^k)比较复杂时很有用,
8     ↪ 但f(p^k)可以递推时就可以省略了
9 // 这里没有记录最小质因子, 但根据线性筛的性质, 每个合数只
10    ↪ 会被它最小的质因子筛掉
11 bool notp[maxn]; // 顾名思义
12
13 void get_table(int n) {
14     sigma_one[1] = 1; // 积性函数必有f(1) = 1
15     for (int i = 2; i <= n; i++) {
16         if (!notp[i]) { // 质数情况
17             prime[++prime[0]] = i;
18             sigma_one[i] = i + 1;
19             f[i] = g[i] = 1;
20         }
21         for (int j = 1; j <= prime[0] && i * prime[j] <= n;
22             ↪ j++) {
23             notp[i * prime[j]] = true;
24
25             if (i % prime[j]) { // 加入一个新的质因子, 这种
26                 ↪ 情况很简单
27                 sigma_one[i * prime[j]] = (long
28                     ↪ long)sigma_one[i] * (prime[j] + 1) % p;
29                 f[i * prime[j]] = i;
30                 g[i * prime[j]] = 1;
31             }
32             else { // 再加入一次最小质因子, 需要再进行分类讨
33                 ↪ 论
34                 f[i * prime[j]] = f[i];
35                 g[i * prime[j]] = g[i] + 1;
36                 // 对于f(p^k)可以直接递推的函数, 这里的判断
37                 ↪ 可以改成
38                 // i / prime[j] % prime[j] != 0, 这样可以省
39                 ↪ 下f[j]的空间,
40                 // 但常数很可能会稍大一些
41
42                 if (f[i] == 1) // 质数的幂次, 这
43                     ↪ 里\sigma_1可以递推
44                 sigma_one[i * prime[j]] = (sigma_one[i]
45                     ↪ + i * prime[j]) % p;
46                 // 对于更一般的情况, 可以借助g[j]计
47                 ↪ 算f(p^k)
48                 else sigma_one[i * prime[j]] = // 否则直接
49                     ↪ 利用积性, 两半乘起来
50                 (long long)sigma_one[i * prime[j] /
51                     ↪ f[i]] * sigma_one[f[i]] % p;
52                 break;
53             }
54         }
55     }
56 }

```

### 2.3 杜教筛

```

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

```

### 2.4 Powerful Number筛

注意Powerful Number筛只能求积性函数的前缀和。本质上就是构造一个方便求前缀和的函数, 然后做类似杜教筛的操作。定义Powerful Number表示每个质因子幂次都大于1的数, 显然最多有 $\sqrt{n}$ 个。设我们要求和的函数是 $f(n)$ , 构造一个方便求前缀和的积性函数 $g(n)$ 使得 $g(p) = f(p)$ 。那么就存在一个积性函数 $h = f * g^{-1}$ , 也就是 $f = g * h$ 。可以证明 $h(p) = 0$ , 所以只有Powerful Number的 $h$ 值不为0。

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

只需要枚举每个Powerful Number作为 $d$ , 然后用杜教筛计算 $g$ 的前缀和。求 $h(d)$ 时要先预处理 $h(p^k)$ , 显然有

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

处理完之后DFS就行了. (显然只需要筛 $\sqrt{n}$ 以内的质数.)  
复杂度取决于杜教筛的复杂度, 特殊题目构造的好也可以做到 $O(\sqrt{n})$ .

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

2.5 洲阁筛

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

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

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

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

边界显然就是自然数幂次和, 转移是

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

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

2. 所有质因子都 $\leq \sqrt{n}$
- 类似的道理, 我们继续DP:  $h_{i,j}$ 表示只含有第 $i$ 到 $m$ 个质数作为质因子的所有数的 $f(i)$ 之和. (这里不需要对每个次幂单独DP了; 另外倒着DP是为了方便卡上限.)  
边界显然是 $h_{m+1,j} = 1$ , 转移是

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

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

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

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

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

```
1 constexpr int maxn = 200005, p = 1000000007;
2
3 long long N, val[maxn]; // 询问的n和存储所有整除结果的表
4 int 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 + 1)
29             ↪ / 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
56     for (int j = 1; j <= val[0]; j++) {
57         dp[j] = (dp[j] - pr[r[j]] + pr[l[j]]) % p;
58
59         dp[j] = (dp[j] + p - 1) % p; // 因为DP数组是有1的,
60             ↪ 但后面计算不应该有1
61     }
62 }
63
64 int calc1(long long n) {
65     get_dp(n, 1, g[0]);
66     get_dp(n, 2, g[1]);
67
68     int ans = 0;
69
70     for (int i = 1; i <= sqrtn; i++)
71         ans = (ans + (long long)f[i] * (g[1][getid(N / i)]
72             ↪ - g[0][getid(N / i)])) % p;
73
74     if (ans < 0)
75         ans += p;
76
77     return ans;
78 }
79
80 int calc2(long long n) {
81     for (int j = 1; j <= val[0]; j++)
82         dp[j] = 1;
83
84     for (int i = 1; i <= prime_cnt; i++)
```

```

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

```

## 2.6 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, 31,
14        ↳ 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范围内的质因数分
3 // 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) { // v用
8     ↳ 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;

```

```
23
24 long long p;
25 do
26     p = Pollards_Rho(n);
27 while (!p); // p是任意一个非平凡因子
28
29 if (p == n) {
30     v.push_back(p); // 说明n本身就是质数
31     return;
32 }
33
34 solve(p, v); // 递归分解两半
35 solve(n / p, v);
36 }
37
38 // Pollard's Rho主过程
39 // 需要使用Miller-Rabin作为子算法
40 // 同时需要调用O(1)快速乘和gcd函数
41 long long Pollards_Rho(long long n) {
42     // assert(n > 1);
43
44     if (Miller_Rabin(n))
45         return n;
46
47     long long c = rand() % (n - 2) + 1, i = 1, k = 2, x =
48         rand() % (n - 3) + 2, u = 2; // 注意这里rand函数需
49         要重定义一下
50     while (true) {
51         i++;
52         x = (mul(x, x, n) + c) % n; // mul是O(1)快速乘函数
53
54         long long g = gcd((u - x + n) % n, n);
55         if (g > 1 && g < n)
56             return g;
57
58         if (u == x)
59             return 0; // 失败, 需要重新调用
60
61         if (i == k) {
62             u = x;
63             k *= 2;
64         }
65     }
66 }
```

## 2.8 快速阶乘算法

参见“数学/多项式”部分.

## 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, b %
8             m, m);
9
10    return solve((a + b * n) / m, (a + b * n) % m, m, b);
11 }
```

## 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 $x = m_1p + a_1 = m_2q + a_2$ , 其中 $p, q$ 是整数, 则有 $m_1p - m_2q = a_2 - a_1$ . 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 $b = m_1p + a_1, M = \text{lcm}(m_1, m_2)$ .

## 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 $a$ 在原根以下的幂次是两两不同的. 一个性质: 如果 $a, b$ 均与 $p$ 互质, 则 $\delta_p(ab) = \delta_p(a)\delta_p(b)$ 的充分必要条件是 $\gcd(\delta_p(a), \delta_p(b)) = 1$ . 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         i += 1
```



```

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

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

### 2.12.3 其他常用公式

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

$$\varphi * I = id$$

$$\mu * id = \varphi$$

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

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

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

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

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

## 3 图论

### 3.1 最小生成树

#### 3.1.1 Boruvka算法

思想：每次选择连接每个连通块的最小边，把连通块缩起来。

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

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

应用：最小异或生成树

#### 3.1.2 动态最小生成树

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 < e.w;
6     }
7 } e[20][maxn], t[maxn]; // 为了便于回滚, 在每层分治存一个副本
8
9 // 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
10 struct A {
11     int id, u, v;
12 } a[maxn];
13
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是并查集
17 // 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 for (int i = 1; i <= n; i++) { // 并查集初始化
22     p[i] = i;
23     size[i] = 1;
24 }
25
26 for (int i = 1; i <= m; i++) { // 读入与预标号
27     scanf("%d%d%d", &e[0][i].u, &e[0][i].v, &e[0][i].w);
28     e[0][i].id = i;
29     id[0][i] = i;
30 }
31
32 for (int i = 1; i <= q; i++) { // 预处理出调用数组
33     scanf("%d%d", &a[i].id, &a[i].v);
34     a[i].u = e[0][a[i].id].w;
35     e[0][a[i].id].w = a[i].v;
36 }

```

```

37
38 for(int i = q; i; i--)
39     e[0][a[i].id].w = a[i].u;
40
41 CDQ(1, q, 0, m, 0); // 这是调用方法
42
43
44 // 分治主过程  $O(n \log^2 n)$ 
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数
65         ↪ 组
66
67     CDQ(l, mid, d + 1, m, ans);
68
69     for (int i = 1; i <= mid; i++)
70         e[d][id[d][a[i].id]].w = a[i].v; // 进行左边的修改
71
72     copy(e[d] + 1, e[d] + m + 1, e[d + 1] + 1);
73     for (int i = 1; i <= m; i++)
74         id[d + 1][e[d][i].id] = i; // 重新准备下一层要用的
75         ↪ 数组
76
77     CDQ(mid + 1, r, d + 1, m, ans);
78
79     for (int i = top; i > tmp; i--)
80         cut(stk[i]); // 撤销所有操作
81     top = tmp;
82 }
83
84 // Reduction(减少无用边): 待修改边标为+INF, 跑MST后把非树边删
85     ↪ 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];
100         }
101
102     for (int i = r; i >= l; i--)
103         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改
104         ↪ 回去

```

```

102     m=cnt;
103 }
104
105 // Contraction(缩必须边):待修改边标为-INF,跑MST后缩除待修改
    ↳ 边之外的所有树边
107 // 返回缩掉的边的总权值
108 // 需要调用Kruskal
109 long long Contraction(int l, int r, int d, int &m) {
110     long long ans = 0;
111
112     for (int i = l; i <= r; i++)
113         e[d][id[d][a[i].id]].w = -INF; // 待修改边标为-INF
114
115     Kruskal(m, e[d]);
116     copy(e[d] + 1, e[d] + m + 1, t + 1);
117
118     int cnt = 0;
119     for (int i = 1; i <= m; i++) {
120
121         if (t[i].w != -INF && t[i].vis) { // 必须边
122             ans += t[i].w;
123             mergeset(t[i].u, t[i].v);
124         }
125         else { // 不确定边
126             id[d][t[i].id]++; cnt++;
127             e[d][cnt] = t[i];
128         }
129     }
130
131     for (int i = r; i >= l; i--) {
132         e[d][id[d][a[i].id]].w = a[i].u; // 把待修改的边改
            ↳ 回去
133         e[d][id[d][a[i].id]].vis = false;
134     }
135
136     m = cnt;
137
138     return ans;
139 }
140
141 // Kruskal算法 O(mLogn)
142 // 方便起见,这里直接沿用进行过缩点的并查集,在过程结束后撤
    ↳ 销即可
144 long long Kruskal(int m, edge *e) {
145     int tmp = top;
146     long long ans = 0;
147
148     sort(e + 1, e + m + 1); // 比较函数在结构体中定义过了
149
150     for (int i = 1; i <= m; i++) {
151         if (findroot(e[i].u) != findroot(e[i].v)) {
152             e[i].vis = true;
153             ans += e[i].w;
154             mergeset(e[i].u, e[i].v);
155         }
156         else
157             e[i].vis = false;
158     }
159
160     for (int i = top; i > tmp; i--)
161         cut(stk[i]); // 撤销所有操作
162     top = tmp;
163
164     return ans;
165 }
166
167 // 以下是并查集相关函数
168

```

```

169 int findroot(int x) { // 因为需要撤销,不写路径压缩
170     while (p[x] != x)
171         x = p[x];
172
173     return x;
174 }
175
176 void mergeset(int x, int y) { // 按size合并,如果想跑得更快
    ↳ 就写一个按秩合并
177     x = findroot(x); // 但是按秩合并要再开一个栈记录合并之
        ↳ 前的秩
178     y = findroot(y);
179
180     if (x == y)
181         return;
182
183     if (size[x] > size[y])
184         swap(x, y);
185
186     p[x] = y;
187     size[y] += size[x];
188     stk[++top] = x;
189 }
190
191 void cut(int x) { // 并查集撤销
192     int y = x;
193
194     do
195         size[y = p[y]] -= size[x];
196     while (p[y] != y);
197
198     p[x] = x;
199 }

```

### 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 = 0x3f3f3f3f;
2
3  struct edge {
4      int u, v, w;
5  } e[maxe];
6
7  int mn[maxn], pr[maxn], ufs[maxn], vis[maxn];
8  bool alive[maxn];
9
10 int edmonds(int n, int m, int rt) {
11     for (int i = 1; i <= n; i++)
12         alive[i] = true;
13
14     int ans = 0;
15
16     while (true) {
17         memset(mn, 63, sizeof(int) * (n + 1));
18         memset(pr, 0, sizeof(int) * (n + 1));
19         memset(ufs, 0, sizeof(int) * (n + 1));
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 < mn[e[i].v]) {
26                 mn[e[i].v] = e[i].w;
27                 pr[e[i].v] = e[i].u;

```

```

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

```

### $O(m \log n)$ 版本

(堆优化版本可以参考fstqwq的模板, 在最后没有目录的部分.)

#### 3.1.4 Steiner Tree 斯坦纳树

问题: 一张图上有 $k$ 个关键点, 求让关键点两两连通的最小生成树

做法: 状压DP,  $f_{i,S}$ 表示以 $i$ 号为点为树根,  $i$ 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); // 不要忘了是
19        // 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 = (t - 1) & s)
33                dp[i][s] = min(dp[i][s], dp[i][t] + dp[i][s ^ t]);
34
35        // SPFA
36        queue<int> q;
37        for (int i = 1; i <= n; i++)
38            if (dp[i][s] < inf) {
39                q.push(i);
40                inq[i] = true;
41            }
42
43        while (!q.empty()) {
44            int i = q.front();
45            q.pop();
46            inq[i] = false; // 最终结束时inq一定全0, 所以不
47            // 用清空
48
49            for (int j = 1; j <= n; j++)
50                if (dp[i][s] + g[i][j] < dp[j][s]) {
51                    dp[j][s] = dp[i][s] + g[i][j];
52                    if (!inq[j]) {
53                        q.push(j);
54                        inq[j] = true;
55                    }
56                }
57        }
58
59        int ans = inf;
60        for (int i = 1; i <= n; i++)
61            ans = min(ans, dp[i][(1 << k) - 1]);
62
63        printf("%d\n", ans);
64
65        return 0;
66    }

```

### 3.1.5 最小直径生成树

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

## 3.2 最短路

### 3.2.1 Dijkstra

见k短路(注意那边是求到 $t$ 的最短路).

### 3.2.2 Johnson算法(负权图多源最短路)

首先前提是图没有负环.

先任选一个起点 $s$ ,跑一边SPFA,计算每个点的势 $h_u = d_{s,u}$ ,然后将每条边 $u \rightarrow v$ 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
  // 最好还是用A*
2 // DAG和有环的情况都可以,有重边或自环也无所谓,但不能有零
  // 环
3 // 以下代码以Dijkstra + 可持久化左偏树为例
4
5 constexpr int maxn = 1005, maxe = 10005, maxm = maxe * 30;
  // 点,边数,左偏树结点数
6
7 // 结构体定义
8 struct A { // 用来求最短路
9     int x, d;
10
11     A(int x, int d) : x(x), d(d) {}
12
13     bool operator < (const A &a) const {
14         return d > a.d;
15     }
16 };
17
18 struct node { // 左偏树结点
19     int w, i, d; // i: 最后一条边的编号 d: 左偏树附加信息
20     node *lc, *rc;
21
22     node() {}
23
24     node(int w, int i) : w(w), i(i), d(0) {}
25
26     void refresh(){
27         d = rc -> d + 1;
28     }
29 } null[maxm], *ptr = null, *root[maxn];
30
31 struct B { // 维护答案用
32     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]; // 最开始要存反向
  // 图,然后把G清空作为儿子列表
```

```
44 bool vis[maxn], used[maxe]; // used表示边是否在最短路树上
45 int u[maxe], v[maxe], w[maxe]; // 存下每条边,注意是有向边
46 int d[maxn], p[maxn]; // p表示最短路树上每个点的父边
47 int n, m, k, s, t; // s, t分别表示起点和终点
48
49 // 以下是主函数中较关键的部分
50 for (int i = 0; i <= n; i++)
51     root[i] = null; // 一定要加上!!!
52
53 // (读入&建反向图)
54
55 Dijkstra();
56
57 // (清空G, W, id)
58
59 for (int i = 1; i <= n; i++)
60     if (p[i]) {
61         used[p[i]] = true; // 在最短路树上
62         G[v[p[i]]].push_back(i);
63     }
64
65 for (int i = 1; i <= m; i++) {
66     w[i] = d[u[i]] - d[v[i]]; // 现在的w[i]表示这条边能使
      // 路径长度增加多少
67     if (!used[i])
68         root[u[i]] = merge(root[u[i]], newnode(w[i], i));
69 }
70
71 dfs(t);
72
73 priority_queue<B> heap;
74 heap.push(B(s, root[s], 0)); // 初始状态是找贡献最小的边加
      // 进去
75
76 printf("%d\n", d[s]); // 第1短路需要特判
77 while (--k) { // 其余k - 1短路径用二叉堆维护
78     if (heap.empty())
79         printf("-1\n");
80     else {
81         int x = heap.top().x, w = heap.top().w;
82         node *rt = heap.top().rt;
83         heap.pop();
84
85         printf("%d\n", d[s] + w + rt -> w);
86
87         if (rt -> lc != null || rt -> rc != null)
88             heap.push(B(x, merge(rt -> lc, rt -> rc), w));
      // pop掉当前边,换成另一条贡献大一点的边
89         if (root[v[rt -> i]] != null)
90             heap.push(B(v[rt -> i], root[v[rt -> i]], w +
      // 保留当前边,往后面再接上另一
      // 条边
91     }
92 }
93 // 主函数到此结束
94
95 // Dijkstra预处理最短路 O(m log n)
96 void Dijkstra() {
97     memset(d, 63, sizeof(d));
98     d[t] = 0;
99     priority_queue<A> heap;
100     heap.push(A(t, 0));
101
102     while (!heap.empty()) {
103         int x = heap.top().x;
104         heap.pop();
105
106         if (vis[x])
107             continue;
```



```

110     vis[x] = true;
111     for (int i = 0; i < (int)G[x].size(); i++)
112         if (!vis[G[x][i]] && d[G[x][i]] > d[x] + W[x]
113             ↳ [i]) {
114             d[G[x][i]] = d[x] + W[x][i];
115             p[G[x][i]] = id[x][i];
116
117             heap.push(A(G[x][i], d[G[x][i]]));
118         }
119     }
120 }
121
122 // dfs求出每个点的堆 总计 $O(m \log n)$ 
123 // 需要调用merge, 同时递归调用自身
124 void dfs(int x) {
125     root[x] = merge(root[x], root[v[p[x]]]);
126
127     for (int i = 0; i < (int)G[x].size(); i++)
128         dfs(G[x][i]);
129 }
130
131 // 包装过的new node()  $O(1)$ 
132 node *newnode(int w, int i) {
133     *++ptr = node(w, i);
134     ptr -> lc = ptr -> rc = null;
135     return ptr;
136 }
137
138 // 带可持久化的左偏树合并 总计 $O(\log n)$ 
139 // 递归调用自身
140 node *merge(node *x, node *y) {
141     if (x == null)
142         return y;
143     if (y == null)
144         return x;
145
146     if (x -> w > y -> w)
147         swap(x, y);
148
149     node *z = newnode(x -> w, x -> i);
150     z -> lc = x -> lc;
151     z -> rc = merge(x -> rc, y);
152
153     if (z -> lc -> d < z -> rc -> d)
154         swap(z -> lc, z -> rc);
155     z -> refresh();
156
157     return z;
158 }

```

### 3.3 Tarjan算法

#### 3.3.1 强连通分量

```

1 int dfn[maxn], low[maxn], tim = 0;
2 vector<int> G[maxn], scc[maxn];
3 int sccid[maxn], scc_cnt = 0, stk[maxn];
4 bool instk[maxn];
5
6 void dfs(int x) {
7     dfn[x] = low[x] = ++tim;
8
9     stk[++stk[0]] = x;
10    instk[x] = true;
11
12    for (int y : G[x]) {
13        if (!dfn[y]) {
14            dfs(y);

```

```

15        low[x] = min(low[x], low[y]);
16    }
17    else if (instk[y])
18        low[x] = min(low[x], dfn[y]);
19    }
20
21    if (dfn[x] == low[x]) {
22        scc_cnt++;
23
24        int u;
25        do {
26            u = stk[stk[0]--];
27            instk[u] = false;
28            sccid[u] = scc_cnt;
29            scc[scc_cnt].push_back(u);
30        } while (u != x);
31    }
32 }
33
34 void tarjan(int n) {
35     for (int i = 1; i <= n; i++)
36         if (!dfn[i])
37             dfs(i);
38 }

```

#### 3.3.2 割点 点双

```

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

```

```

42     low[x] = min(low[x], dfn[y]);
43 }
44 }
45
46 if (!pr && child == 1)
47     iscut[x] = false;
48 }
49
50 void Tarjan(int n) {
51     for (int i = 1; i <= n; i++)
52         if (!dfn[i])
53             dfs(i, 0);
54 }

```

### 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
7 vector<long long> d[maxn];
8
9 stack<int> stk;

```

```

10
11 int p[maxn];
12
13 bool vis[maxn], vise[maxn * 2];
14
15 int last[maxn], cnte;
16
17 long long f[maxn], g[maxn], sum[maxn];
18
19 int n, m, cnt;
20
21 void addedge(int x, int y, int w) {
22     v[x].push_back(make_pair(y, w));
23 }
24
25 void dfs(int x) {
26
27     vis[x] = true;
28
29     for (int i = last[x]; ~i; i = e[i].prev) {
30         if (vise[i ^ 1])
31             continue;
32
33         int y = e[i].to, w = e[i].w;
34
35         vise[i] = true;
36
37         if (!vis[y]) {
38             stk.push(i);
39             p[y] = x;
40             dfs(y);
41
42             if (!stk.empty() && stk.top() == i) {
43                 stk.pop();
44                 addedge(x, y, w);
45             }
46         }
47         else {
48             cnt++;
49
50             long long tmp = w;
51             while (!stk.empty()) {
52                 int i = stk.top();
53                 stk.pop();
54
55                 int yy = e[i].to, ww = e[i].w;
56
57                 addedge(cnt, yy, 0);
58
59                 d[cnt].push_back(tmp);
60
61                 tmp += ww;
62
63                 if (e[i ^ 1].to == y)
64                     break;
65             }
66
67             addedge(y, cnt, 0);
68
69             sum[cnt] = tmp;
70
71         }
72     }
73 }
74
75 void dp(int x) {
76
77     for (auto o : v[x]) {
78         int y = o.first, w = o.second;
79         dp(y);

```

```

80     }
81
82     if (x <= n) {
83         for (auto o : v[x]) {
84             int y = o.first, w = o.second;
85
86             f[x] += 2 * w + f[y];
87         }
88
89         g[x] = f[x];
90
91         for (auto o : v[x]) {
92             int y = o.first, w = o.second;
93
94             g[x] = min(g[x], f[x] - f[y] - 2 * w + g[y] +
95                 ↪ w);
96         }
97     }
98     f[x] = sum[x];
99     for (auto o : v[x]) {
100         int y = o.first;
101
102         f[x] += f[y];
103     }
104
105     g[x] = f[x];
106
107     for (int i = 0; i < (int)v[x].size(); i++) {
108         int y = v[x][i].first;
109
110         g[x] = min(g[x], f[x] - f[y] + g[y] + min(d[x]
111             ↪ [i], sum[x] - d[x][i]));
112     }
113 }

```

## 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                 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
31     return ans;
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: 右边
4     ↪ 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                 }
36                 else
37                     flag = true;
38             }
39
40     return flag;
41 }
42
43 bool dfs(int x) {
44     for (int y : G[x])
45         if (!vis[y] && dy[y] == dx[x] + 1) {
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     return false;
56 }
57
58 int hopcroft_karp(int n) {

```

```

60 int ans = 0;
61
62 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
63     for (int y : G[x])
64         if (!boy[y]) {
65             girl[x] = y;
66             boy[y] = x;
67             ans++;
68             break;
69         }
70
71 while (bfs(n)) {
72     memset(vis, 0, sizeof(bool) * (n + 1));
73
74     for (int x = 1; x <= n; x++)
75         if (!girl[x])
76             ans += dfs(x);
77 }
78
79 return ans;
80 }

```

### 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
28     return true;
29 }
30
31 // bfs每个点
32 void bfs(int x) {
33     memset(q, 0, sizeof(q));
34     head = tail = 0;
35
36     q[tail++] = x;
37     visx[x] = true;
38
39     while (true) {
40         while (head != tail) {
41             int x = q[head++];

```

```

43         for (int y = 1; y <= N; y++)
44             if (!visy[y]) {
45                 long long d = lx[x] + ly[y] - w[x][y];
46
47                 if (d < slack[y]) {
48                     p[y] = x;
49                     slack[y] = d;
50
51                     if (!slack[y] && check(y))
52                         return;
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
95     long long ans = 0;
96     for (int i = 1; i <= N; i++)
97         ans += w[i][girl[i]];
98     return ans;
99 }
100
101 // 为了方便贴上主函数
102 int main() {
103
104     scanf("%d%d", &n, &m, &e);
105     N = max(n, m);
106
107     while (e--) {
108         int x, y, c;
109         scanf("%d%d", &x, &y, &c);
110         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, 也
2 // 更方便进行可行边的判定
3 // 注意这个算法复杂度是满的, 并且常数有点大, 而带花树通常
4 // 是跑不满的
5 // 以及, 根据Tutte定理, 如果求最大匹配的大小的话直接输
6 // 出Tutte矩阵的秩/2即可
7 // 需要输出方案时才需要再写后面那些乱七八糟的东西
8
9 // 复杂度和常数所限, 1s之内500已经是这个算法的极限了
10 const int maxn = 505, p = 1000000007; // p可以是任意10^9以
11 // 内的质数
12
13 // 全局数组和变量定义
14 int A[maxn][maxn], B[maxn][maxn], t[maxn][maxn], id[maxn],
15 // a[maxn];
16 bool row[maxn] = {false}, col[maxn] = {false};
17 int n, m, girl[maxn]; // girl是匹配点, 用来输出方案
18
19 // 为了方便使用, 贴上主函数
20 // 需要调用高斯消元和eliminate
21 int main() {
22     srand(19260817);

```

```

19 scanf("%d%d", &n, &m); // 点数和边数
20 while (m--) {
21     int x, y;
22     scanf("%d%d", &x, &y);
23     A[x][y] = rand() % p;
24     A[y][x] = -A[x][y]; // Tutte矩阵是反对称矩阵
25 }
26
27 for (int i = 1; i <= n; i++)
28     id[i] = i; // 输出方案用的, 因为高斯消元的时候会交
29     // 换列
30 memcpy(t, A, sizeof(t));
31 Gauss(A, NULL, n);
32
33 m = n;
34 n = 0; // 这里变量复用纯属个人习惯
35
36 for (int i = 1; i <= m; i++)
37     if (A[id[i]][id[i]])
38         a[++n] = i; // 找出一个极大满秩子矩阵
39
40 for (int i = 1; i <= n; i++)
41     for (int j = 1; j <= n; j++)
42         A[i][j] = t[a[i]][a[j]];
43
44 Gauss(A, B, n);
45
46 for (int i = 1; i <= n; i++)
47     if (!girl[a[i]])
48         for (int j = i + 1; j <= n; j++)
49             if (!girl[a[j]] && t[a[i]][a[j]] && B[j]
50                 // 注意上面那句if的写法, 现在t是邻接矩
51                 // 阵的备份,
52                 // 逆矩阵j行i列不为0当且仅当这条边可行
53                 girl[a[i]] = a[j];
54                 girl[a[j]] = a[i];
55
56                 eliminate(i, j);
57                 eliminate(j, i);
58                 break;
59 }
60
61 printf("%d\n", n / 2);
62 for (int i = 1; i <= m; i++)
63     printf("%d ", girl[i]);
64
65 return 0;
66 }
67 // 高斯消元 O(n^3)
68 // 在传入B时表示计算逆矩阵, 传入NULL则只需计算矩阵的秩
69 void Gauss(int A[][maxn], int B[][maxn], int n) {
70     if (B) {
71         memset(B, 0, sizeof(t));
72         for (int i = 1; i <= n; i++)
73             B[i][i] = 1;
74     }
75
76     for (int i = 1; i <= n; i++) {
77         if (!A[i][i]) {
78             for (int j = i + 1; j <= n; j++)
79                 if (A[j][i]) {
80                     swap(id[i], id[j]);
81                     for (int k = i; k <= n; k++)
82                         swap(A[i][k], A[j][k]);
83
84                     if (B)

```



```

85         for (int k = 1; k <= n; k++)
86             swap(B[i][k], B[j][k]);
87         break;
88     }
89
90     if (!A[i][i])
91         continue;
92 }
93
94 int inv = qpow(A[i][i], p - 2);
95
96 for (int j = 1; j <= n; j++)
97     if (i != j && A[j][i]){
98         int t = (long long)A[j][i] * inv % p;
99
100        for (int k = i; k <= n; k++)
101            if (A[i][k])
102                A[j][k] = (A[j][k] - (long long)t *
103                    ↪ A[i][k]) % p;
104
105        if (B)
106            for (int k = 1; k <= n; k++)
107                if (B[i][k])
108                    B[j][k] = (B[j][k] - (long
109                        ↪ long)t * B[i][k]) % p;
110    }
111
112    if (B)
113        for (int i = 1; i <= n; i++) {
114            int inv = qpow(A[i][i], p - 2);
115
116            for (int j = 1; j <= n; j++)
117                if (B[i][j])
118                    B[i][j] = (long long)B[i][j] * inv % p;
119        }
120
121    // 消去一行一列  $O(n^2)$ 
122    void eliminate(int r, int c) {
123        row[r] = col[c] = true; // 已经被消掉
124
125        int inv = qpow(B[r][c], p - 2);
126
127        for (int i = 1; i <= n; i++)
128            if (!row[i] && B[i][c]) {
129                int t = (long long)B[i][c] * inv % p;
130
131                for (int j = 1; j <= n; j++)
132                    if (!col[j] && B[r][j])
133                        B[i][j] = (B[i][j] - (long long)t *
134                            ↪ B[r][j]) % p;
135            }
136    }

```

### 3.6.2 带花树

```

1 // 带花树通常比高斯消元快很多, 但在只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], tim,
8   ↪ q[maxn], head, tail;
9 int n, m;
10
11 // 封装好的主过程  $O(nm)$ 

```

```

11 int blossom() {
12     int ans = 0;
13
14     for (int i = 1; i <= n; i++)
15         if (!girl[i])
16             ans += bfs(i);
17
18     return ans;
19 }
20
21 // bfs找增广路  $O(m)$ 
22 bool bfs(int s) {
23     memset(t, 0, sizeof(t));
24     memset(p, 0, sizeof(p));
25
26     for (int i = 1; i <= n; i++)
27         f[i] = i; // 并查集
28
29     head = tail = 0;
30     q[tail++] = s;
31     t[s] = 1;
32
33     while (head != tail) {
34         int x = q[head++];
35         for (int y : G[x]) {
36             if (findroot(y) == findroot(x) || t[y] == 2)
37                 continue;
38
39             if (!t[y]) {
40                 t[y] = 2;
41                 p[y] = x;
42
43                 if (!girl[y]) {
44                     for (int u = y, t; u = t) {
45                         t = girl[p[u]];
46                         girl[p[u]] = u;
47                         girl[u] = p[u];
48                     }
49                     return true;
50                 }
51
52                 t[girl[y]] = 1;
53                 q[tail++] = girl[y];
54             }
55             else if (t[y] == 1) {
56                 int z = LCA(x, y);
57
58                 shrink(x, y, z);
59                 shrink(y, x, z);
60             }
61         }
62     }
63
64     return false;
65 }
66
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)

```

```

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

```

### 3.6.3 带权带花树

Forked from templates 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 - g[u]
  ↳ [v].w when minimum weight blossom is needed
2 //type of ans is Long Long
3 //replace all int to Long Long if weight of edge is Long
  ↳ Long
4
5 struct WeightGraph {
6     static const int INF = INT_MAX;
7     static const int MAXN = 400;
8     struct edge{
9         int u, v, w;
10        edge() {}
11        edge(int u, int v, int w): u(u), v(v), w(w) {}
12    };
13    int n, n_x;
14    edge g[MAXN * 2 + 1][MAXN * 2 + 1];
15    int lab[MAXN * 2 + 1];
16    int match[MAXN * 2 + 1], slack[MAXN * 2 + 1], st[MAXN *
  ↳ 2 + 1], pa[MAXN * 2 + 1];
17    int flower_from[MAXN * 2 + 1][MAXN+1], S[MAXN * 2 + 1],
  ↳ vis[MAXN * 2 + 1];
18    vector<int> flower[MAXN * 2 + 1];
19    queue<int> q;
20    inline int e_delta(const edge &e){ // does not work
  ↳ inside blossoms
21        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
22    }
23    inline void update_slack(int u, int x){
24        if(!slack[x] || e_delta(g[u][x]) <
  ↳ e_delta(g[slack[x]][x]))
25            slack[x] = u;
26    }
27    inline void set_slack(int x){

```

```

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

```

```

91     flower[b].push_back(y = st[match[x]]),
92     q_push(y);
93 }
94 reverse(flower[b].begin() + 1, flower[b].end());
95 for(int x = v, y; x != lca; x = st[pa[y]]) {
96     flower[b].push_back(x),
97     flower[b].push_back(y = st[match[x]]),
98     q_push(y);
99 }
100 set_st(b, b);
101 for(int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w
    ⇨ 0;
102 for(int x = 1; x <= n; ++x) flower_from[b][x] = 0;
103 for(size_t i = 0; i < flower[b].size(); ++i){
104     int xs = flower[b][i];
105     for(int x = 1; x <= n_x; ++x)
106         if(g[b][x].w == 0 || e_delta(g[xs][x]) <
            ⇨ e_delta(g[b][x]))
107             g[b][x] = g[xs][x], g[x][b] = g[x][xs];
108     for(int x = 1; x <= n; ++x)
109         if(flower_from[xs][x]) flower_from[b][x] =
            ⇨ xs;
110 }
111 set_slack(b);
112 }
113 inline void expand_blossom(int b){ // S[b] == 1
114     for(size_t i = 0; i < flower[b].size(); ++i)
115         set_st(flower[b][i], flower[b][i]);
116     int xr = flower_from[b][g[b][pa[b]].u], pr =
        ⇨ get_pr(b, xr);
117     for(int i = 0; i < pr; i += 2){
118         int xs = flower[b][i], xns = flower[b][i + 1];
119         pa[xs] = g[xns][xs].u;
120         S[xs] = 1, S[xns] = 0;
121         slack[xs] = 0, set_slack(xns);
122         q_push(xns);
123     }
124     S[xr] = 1, pa[xr] = pa[b];
125     for(size_t i = pr + 1; i < flower[b].size(); ++i){
126         int xs = flower[b][i];
127         S[xs] = -1, set_slack(xs);
128     }
129     st[b] = 0;
130 }
131 inline bool on_found_edge(const edge &e){
132     int u = st[e.u], v = st[e.v];
133     if(S[v] == -1){
134         pa[v] = e.u, S[v] = 1;
135         int nu = st[match[v]];
136         slack[v] = slack[nu] = 0;
137         S[nu] = 0, q_push(nu);
138     }else if(S[v] == 0){
139         int lca = get_lca(u, v);
140         if(!lca) return augment(u, v), augment(v, u),
            ⇨ true;
141         else add_blossom(u, lca, v);
142     }
143     return false;
144 }
145 inline bool matching(){
146     memset(S + 1, -1, sizeof(int) * n_x);
147     memset(slack + 1, 0, sizeof(int) * n_x);
148     q = queue<int>();
149     for(int x = 1; x <= n_x; ++x)
150         if(st[x] == x && !match[x]) pa[x]=0, S[x]=0,
            ⇨ q_push(x);
151     if(q.empty())return false;
152     for(;;){
153         while(q.size()){

```

```

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

```

```

214     }
215 };

```

### 3.6.4 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, son存
   ↳ 的是sdom树上的儿子
2
3 int ufs[maxn];
4
5 int idom[maxn], sdom[maxn], anc[maxn]; // anc: sdom的dfn最
   ↳ 小的祖先
6
7 int p[maxn], dfn[maxn], id[maxn], tim;
8
9 int findufs(int x) {
10     if (ufs[x] == x)
11         return x;
12
13     int t = ufs[x];
14     ufs[x] = findufs(ufs[x]);
15
16     if (dfn[sdom[anc[x]]] > dfn[sdom[anc[t]]])
17         anc[x] = anc[t];
18
19     return ufs[x];
20 }
21
22 void dfs(int x) {
23     dfn[x] = ++tim;
24     id[tim] = x;
25     sdom[x] = x;
26
27     for (int y : G[x])
28         if (!dfn[y]) {
29             p[y] = x;
30             dfs(y);
31         }
32 }
33
34 void get_dominator(int n) {
35     for (int i = 1; i <= n; i++)
36         ufs[i] = anc[i] = i;
37

```

```

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

```

## 3.8 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], i <
   ↳ n ? ' ' : '\n');
8 }

```

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

```

1 bool vis[maxn];
2 int stk[maxn], top;
3
4 // 主函数
5 for (int i = 0; i < n; i += 2)
6     if (!vis[i] && !vis[i ^ 1]) {
7         top = 0;
8         if (!dfs(i)) {
9             while (top)
10                 vis[stk[top--]] = false;
11
12         if (!dfs(i + 1)) {

```

```

13         bad = true;
14         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 }

```

## 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更优,稠
  // 密图则HLPP更优
2
3 struct edge{
4     int to, cap, prev;
5 } e[maxe * 2];
6
7 int last[maxn], len, d[maxn], cur[maxn], q[maxn];
8
9 // main函数里要初始化
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 && (f =
47             ↪ dfs(e[i].to, min(e[i].cap, a)))) {
48
49                 e[i].cap -= f;
50                 e[i^1].cap += f;
51                 flow += f;
52                 a -= f;
53
54                 if (!a)
55                     break;
56
57     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 }

```

### 3.9.2 ISAP

可能有毒, 慎用.

```

1 // 注意ISAP适用于一般稀疏图,对于二分图或分层图情况Dinic比
  // 较优, 稠密图则HLPP更优
2
3 // 边的定义
4 // 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],
  // ↪ cur[maxn], q[maxn];
11 int n, m, s, t; // s, t一定要开成全局变量
12
13 void AddEdge(int x, int y, int z) {
14     e[cnte].to = y;
15     e[cnte].cap = z;
16     e[cnte].prev = last[x];
17     last[x] = cnte++;
18 }
19
20 void addedge(int x, int y, int z) {
21     AddEdge(x, y, z);
22     AddEdge(y, x, 0);
23 }
24
25 // 预处理到t的距离标号
26 // 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
27 void bfs() {
28     memset(d, -1, sizeof(d));
29
30     int head = 0, tail = 0;
31     d[t] = 0;
32     q[tail++] = t;
33

```

```

34     while (head != tail) {
35         int x = q[head++];
36         c[d[x]]++;
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的时候一定要开个变
63 // 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, 并返回s重新
73             // 找增广路
74             flow += augment();
75             x = s;
76         }
77
78         bool ok = false;
79         for (int &i = cur[x]; ~i; i = e[i].prev)
80             if (e[i].cap && d[x] == d[e[i].to] + 1) {
81                 p[e[i].to] = i;
82                 x = e[i].to;
83             }
84
85         ok = true;
86         break;
87     }
88
89     if (!ok) { // 修改距离标号
90         int tmp = n - 1;
91         for (int i = last[x]; ~i; i = e[i].prev)
92             if (e[i].cap)
93                 tmp = min(tmp, d[e[i].to] + 1);
94
95         if (!--c[d[x]])
96             break; // gap优化, 一定要加上
97
98         c[d[x]] = tmp++;
99         cur[x] = last[x];
100
101         if (x != s)
102             x = e[p[x] ^ 1].to;
103     }
104 }

```

```

102     return flow;
103 }
104
105 // 重要! main函数最前面一定要加上如下初始化
106 memset(last, -1, sizeof(last));

```

### 3.9.3 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, 记得改这
37     // 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                 h[e[i].to] = h[x] + 1;
47                 q[tail++] = e[i].to;
48             }
49     }
50
51     return h[s] < 2 * n;
52 }
53
54 void push(int x) {
55     for (int i = last[x]; ~i; i = e[i].prev)
56         if (e[i].cap && h[x] == h[e[i].to] + 1) {
57             int d = min(ex[x], (long long)e[i].cap);
58
59             e[i].cap -= d;
60             e[i ^ 1].cap += d;
61             ex[x] -= d;

```



```

61     ex[e[i].to] += d;
62
63     if (e[i].to != s && e[i].to != t &&
        ↪ !inq[e[i].to]) {
64         heap.push(e[i].to);
65         inq[e[i].to] = true;
66     }
67
68     if (!ex[x])
69         break;
70 }
71 }
72
73 void relabel(int x) {
74     h[x] = 2 * n;
75
76     for (int i = last[x]; ~i; i = e[i].prev)
77         if (e[i].cap)
78             h[x] = min(h[x], h[e[i].to] + 1);
79 }
80
81 long long hlpp() {
82     if (!bfs())
83         return 0;
84
85     // memset(gap, 0, sizeof(int) * 2 * n);
86     h[s] = n;
87
88     for (int i = 1; i <= n; i++)
89         gap[h[i]]++;
90
91     for (int i = last[s]; ~i; i = e[i].prev)
92         if (e[i].cap) {
93             int d = e[i].cap;
94
95             e[i].cap -= d;
96             e[i ^ 1].cap += d;
97             ex[s] -= d;
98             ex[e[i].to] += d;
99
100             if (e[i].to != s && e[i].to != t &&
                ↪ !inq[e[i].to]) {
101                 heap.push(e[i].to);
102                 inq[e[i].to] = true;
103             }
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] > h[x])
116                         h[i] = n + 1;
117             }
118
119             relabel(x);
120             ++gap[h[x]];
121             heap.push(x);
122             inq[x] = true;
123         }
124     }
125
126     return ex[t];
127 }

```

```

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

```

```

63     e[cnte].w = w;
64
65     e[cnte].prev = last[x];
66     last[x] = cnte++;
67 }
68
69 void addedge(int x, int y, int c, int w) {
70     add(x, y, c, w);
71     add(y, x, 0, -w);
72 }

```

### 3.10.2 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
21 void addedge(int x, int y, int z, int w) {
22     Adde(x, y, z, w);
23     Adde(y, x, 0, -w);
24 }
25
26 void dijkstra() {
27     memset(d, 63, sizeof(d));
28     memset(vis, 0, sizeof(vis));
29
30     priority_queue<pair<long long, int>> heap;
31
32     d[s] = 0;
33     heap.push(make_pair(0ll, s));
34
35     while (!heap.empty()) {
36         int x = heap.top().second;
37         heap.pop();
38
39         if (vis[x])
40             continue;
41
42         vis[x] = true;
43         for (int i = last[x]; ~i; i = e[i].prev)
44             if (e[i].cap > 0 && d[e[i].to] > d[x] + e[i].w

```

```

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], e[i].to));
49     }
50 }
51
52 pair<long long, long long> mcmf() {
53     /*
54     spfa();
55     for (int i = 1; i <= t; i++)
56         h[i] = d[i];
57     // 如果初始有负权就像这样跑一遍SPFA预处理
58     */
59
60     long long flow = 0, cost = 0;
61
62     while (dijkstra(), d[t] < 0x3f3f3f3f) {
63         for (int i = 1; i <= t; i++)
64             h[i] += d[i];
65
66         int a = 0x3f3f3f3f;
67
68         for (int x = t; x != s; x = e[p[x] ^ 1].to)
69             a = min(a, e[p[x]].cap);
70
71         flow += a;
72         cost += (long long)a * h[t];
73
74         for (int x = t; x != s; x = e[p[x] ^ 1].to) {
75             e[p[x]].cap -= a;
76             e[p[x] ^ 1].cap += a;
77         }
78     }
79
80     return make_pair(flow, cost);
81 }
82
83 // 记得初始化
84 memset(last, -1, sizeof(last));
85

```

## 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。

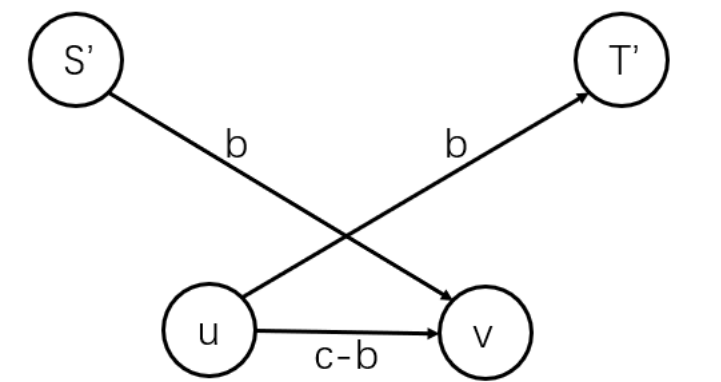
如果一条边是可行边，我们就需要把它删掉，同时进行退流， $u \rightarrow s$ 和 $t \rightarrow v$ 都退掉等同于这条边容量的流量。退流用Dinic实现即可。

3.11.3 费用流

3.11.4 上下界网络流

有源汇上下界最大流

新建超级源汇 $S', T'$ ，然后如图所示转化每一条边。



然后从 $S'$ 到 $S$ ，从 $T$ 到 $T'$ 分别连容量为正无穷的边即可。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。

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。

无源汇上下界可行流

转化方法和上面的图是一样的，只不过不需要考虑原有的源汇了。在新图跑一遍最大流之后检查一遍辅助边，如果有辅助边没满流则无解，否则把每条边的流量加上 $b$ 就是一组可行方案。

3.11.5 常见建图方法

• 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。

- 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 $\infty$ 边，然后 $s$ 向它连 $\infty$ 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。
- 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。
- 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。

3.11.6 例题

- 最大流
- 最小割

1. 切糕

• 费用流

- 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 $k$ 。

- 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。（当然也可以做无根树哈希。）
- 每个点在Prufer序列中出现的次数恰好是度数 $-1$ 。所以如果给定某些点的度数然后求方案数，就可以用简单的组合数解决。

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

线性求Prufer序列代码：

```
1 // 0-based
2 vector<vector<int>> adj;
3 vector<int> parent;
4
5 void dfs(int v) {
6     for (int u : adj[v]) {
7         if (u != parent[v]) parent[u] = v, dfs(u);
8     }
9 }
10
11 vector<int> pruefer_code() { // pruefer是德语
12     int n = adj.size();
13     parent.resize(n), parent[n - 1] = -1;
14     dfs(n - 1);
15
16     int ptr = -1;
17     vector<int> degree(n);
18     for (int i = 0; i < n; i++) {
19         degree[i] = adj[i].size();
20         if (degree[i] == 1 && ptr == -1) ptr = i;
21     }
22
23     vector<int> code(n - 2);
24     int leaf = ptr;
25     for (int i = 0; i < n - 2; i++) {
26         int next = parent[leaf];
27         code[i] = next;
28         if (--degree[next] == 1 && next < ptr)
29             leaf = next;
30         else {
31             ptr++;
32             while (degree[ptr] != 1)
33                 ptr++;
34             leaf = ptr;
35         }
36     }
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;
```

```
4     vector<int> degree(n, 1);
5     for (int i : code) degree[i]++;
6
7     int ptr = 0;
8     while (degree[ptr] != 1) ptr++;
9     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) ptr++;
19             leaf = ptr;
20         }
21     }
22     edges.emplace_back(leaf, n - 1);
23     return edges;
24 }
```

### 3.13 弦图相关

Forked from templates of NEW CODE!!.

1. 团数  $\leq$  色数，弦图团数 = 色数
2. 设  $next(v)$  表示  $N(v)$  中最前的点。令  $w^*$  表示所有满足  $A \in B$  的  $w$  中最后的一个点，判断  $v \cup N(v)$  是否为极大团，只需判断是否存在一个  $w$ ，满足  $Next(w) = v$  且  $|N(v)| + 1 \leq |N(w)|$  即可。
3. 最小染色：完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色
4. 最大独立集：完美消除序列从前往后能选就选
5. 弦图最大独立集数 = 最小团覆盖数，最小团覆盖：设最大独立集为  $\{p_1, p_2, \dots, p_t\}$ ，则  $\{p_1 \cup N(p_1), \dots, p_t \cup N(p_t)\}$  为最小团覆盖

## 4 数据结构

### 4.1 线段树

#### 4.1.1 非递归线段树

让fstqwq手撕

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

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

区间修改要标记永久化,并且求区间和和求最值的代码不太一样

#### 区间加, 区间求和

```
1 void update(int l, int r, int d) {
2     int len = 1, cntl = 0, cntr = 0; // cntl, cntr是左右两
3     // 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 >>= 1, len <= 1) {
5         tree[l] += cntl * d, tree[r] += cntr * d;
6         if (~l & 1) tree[l ^ 1] += d * len, mark[l ^ 1] += d, cntl += len;
7         if (r & 1) tree[r ^ 1] += d * len, mark[r ^ 1] += d, cntr += len;
8     }
9
10    for (; l; l >>= 1, r >>= 1)
11        tree[l] += cntl * d, tree[r] += cntr * d;
12 }
13
14 int query(int l, int r) {
15     int ans = 0, len = 1, cntl = 0, cntr = 0;
16     for (l += n - 1, r += n + 1; l ^ r ^ 1; l >>= 1, r >>= 1, len <= 1) {
17         ans += cntl * mark[l] + cntr * mark[r];
18         if (~l & 1) ans += tree[l ^ 1], cntl += len;
19         if (r & 1) ans += tree[r ^ 1], cntr += len;
20     }
21
22    for (; l; l >>= 1, r >>= 1)
23        ans += cntl * mark[l] + cntr * mark[r];
24
25    return ans;
26 }
```

#### 区间加, 区间求最大值

```
1 void update(int l, int r, int d) {
2     for (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
3         if (l < N) {
```

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

#### 4.1.2 线段树维护矩形并

为线段树的每个结点维护 $cover_i$ 表示这个区间被完全覆盖的次数。更新时分情况讨论, 如果当前区间已被完全覆盖则长度就是区间长度, 否则长度是左右儿子相加。

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

```

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

```

## 4.2 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 = 1;
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 + 1, b + r + 1, a + 1); // 后面的分治会破坏排序,
    // 所以要复制一份
36
37    CDQ2(l, r);
38 }
39
40 void CDQ2(int l, int r) {
41     if (l >= r)
42         return;
43
44     int mid = (l + r) / 2;
45
46     CDQ2(l, mid);
47     CDQ2(mid + 1, r);
48
49     int i = l, j = mid + 1, k = 1;
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 + 1, t + r + 1, b + 1);
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询问的 $k$ 修改一下. 否则需要维护一个全局的数据结构, 一般来说可以先递归右边再递归左边, 具体维护方法视情况而定.

以下代码以ZJOI K大数查询为例(区间都添加一个数, 询问区间 $k$ 大数).

```
1 int op[maxn], ql[maxn], qr[maxn]; // 1: modify 2: query
2 long long qk[maxn]; // 修改和询问可以一起存
3
4 int ans[maxn];
5
6 void solve(int l, int r, vector<int> v) { // 如果想卡常可以
    // 用数组, 然后只需要传一个数组的l, r, 递归的时候类似归并
    // 反过来, 开两个辅助数组, 处理完再复制回去即可
7     if (v.empty())
8         return;
9
10    if (l == r) {
11        for (int i : v)
12            if (op[i] == 2)
13                ans[i] = 1;
14
15        return;
16    }
17
18    int mid = (l + r) / 2;
19
20    vector<int> vl, vr;
21
22    for (int i : v) {
23        if (op[i] == 1) {
24            if (qk[i] <= mid)
25                vl.push_back(i);
26            else {
27                update(ql[i], qr[i], 1); // update是区间加
28                vr.push_back(i);
29            }
30        }
31        else {
32            long long tmp = query(ql[i], qr[i]);
33
34            if (qk[i] <= tmp) // 因为是k大数查询
35                vr.push_back(i);
```

```

36         else {
37             qk[i] -= tmp;
38             vl.push_back(i);
39         }
40     }
41 }
42
43 for (int i : vr)
44     if (op[i] == 1)
45         update(ql[i], qr[i], -1);
46
47 v.clear();
48
49 solve(l, mid, vl);
50 solve(mid + 1, r, vr);
51 }
52
53 int main() {
54     int n, m;
55     scanf("%d%d", &n, &m);
56
57     M = 1;
58     while (M < n + 2)
59         M *= 2;
60
61     for (int i = 1; i <= m; i++)
62         scanf("%d%d%d%lld", &op[i], &ql[i], &qr[i],
63             ↪ &qk[i]);
64
65     vector<int> v;
66     for (int i = 1; i <= m; i++)
67         v.push_back(i);
68
69     solve(1, 1e9, v);
70
71     for (int i = 1; i <= m; i++)
72         if (op[i] == 2)
73             printf("%d\n", ans[i]);
74
75     return 0;
76 }

```

## 4.4 平衡树

pb\_ds平衡树在杂项(倒数第二章)里.

### 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, 优先级
5     node *ch[2]; // 0表示左儿子, 1表示右儿子
6
7     node(int key = 0) : key(key), size(1), p(rand()) {}
8
9     void refresh() {
10         size = ch[0] -> size + ch[1] -> size + 1;
11     } // 更新子树大小(和附加信息, 如果有的话)
12 } null[maxn], *root = null, *ptr = null; // 数组名叫
13     ↪ 做null是为了方便开哨兵节点
14 // 如果需要删除而空间不能直接开下所有结点, 则需要再写一个
15     ↪ 垃圾回收
16 // 注意: 数组里的元素一定不能delete, 否则会导致RE
17
18 // 重要!在主函数最开始一定要加上以下预处理:
19 null -> ch[0] = null -> ch[1] = null;
20 null -> size = 0;

```

```

20 // 伪构造函数 O(1)
21 // 为了方便, 在结点类外面再定义一个伪构造函数
22 node *newnode(int x) { // 键值为x
23     *++ptr = node(x);
24     ptr -> ch[0] = ptr -> ch[1] = null;
25     return ptr;
26 }
27
28 // 插入键值 期望O(\log n)
29 // 需要调用旋转
30 void insert(int x, node *&rt) { // rt为当前结点, 建议调用时
31     ↪ 传入root, 下同
32     if (rt == null) {
33         rt = newnode(x);
34         return;
35     }
36
37     int d = x > rt -> key;
38     insert(x, rt -> ch[d]);
39     rt -> refresh();
40
41     if (rt -> ch[d] -> p < rt -> p)
42         rot(rt, d ^ 1);
43 }
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             int d = rt -> ch[0] -> p < rt -> ch[1] -> p;
52             rot(rt, d);
53             erase(x, rt -> ch[d]);
54         }
55         else
56             rt = rt -> ch[rt -> ch[0] == null];
57     }
58     else
59         erase(x, rt -> ch[x > rt -> key]);
60
61     if (rt != null)
62         rt -> refresh();
63 }
64
65 // 求元素的排名(严格小于键值的个数 + 1) 期望O(\log n)
66 // 非递归
67 int rank(int x, node *rt) {
68     int ans = 1, d;
69     while (rt != null) {
70         if ((d = x > rt -> key))
71             ans += rt -> ch[0] -> size + 1;
72         rt = rt -> ch[d];
73     }
74     return ans;
75 }
76
77 // 返回排名第k(从1开始)的键值对应的指针 期望O(\log n)
78 // 非递归
79 node *kth(int x, node *rt) {
80     int d;
81     while (rt != null) {
82         if (x == rt -> ch[0] -> size + 1)
83             return rt;
84         if ((d = x > rt -> ch[0] -> size))
85             x -= rt -> ch[0] -> size + 1;
86         rt = rt -> ch[d];
87     }
88 }

```

```

89     rt = rt -> ch[d];
90 }
91
92     return rt;
93 }
94
95 // 返回前驱(最大的比给定键值小的键值)对应的指针 期望 $O(\log n)$ 
96 // 非递归
97 node *pred(int x, node *rt) {
98     node *y = null;
99     int d;
100
101     while (rt != null) {
102         if ((d = x > rt -> key))
103             y = rt;
104
105         rt = rt -> ch[d];
106     }
107
108     return y;
109 }
110
111 // 返回后继(最小的比给定键值大的键值)对应的指针 期望 $O(\log n)$ 
112 // 非递归
113 node *succ(int x, node *rt) {
114     node *y = null;
115     int d;
116
117     while (rt != null) {
118         if ((d = x < rt -> key))
119             y = rt;
120
121         rt = rt -> ch[d ^ 1];
122     }
123
124     return y;
125 }
126
127 // 旋转(Treap版本)  $O(1)$ 
128 // 平衡树基础操作
129 // 要求对应儿子必须存在, 否则会导致后续各种莫名其妙的问题
130 // 以维护树结构
131 void rot(node *&x, int d) { // x为被转下去的结点, 会被修改
132     node *y = x -> ch[d ^ 1];
133
134     x -> ch[d ^ 1] = y -> ch[d];
135     y -> ch[d] = x;
136
137     x -> refresh();
138     (x = y) -> refresh();
139 }

```

#### 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];

```

```

14 node *copied(node *x) { // 如果不用可持久化的话, 直接用就行
15     ↪ 了
16     return new node(*x);
17 }
18
19 node *merge(node *x, node *y) {
20     if (x == null)
21         return y;
22     if (y == null)
23         return x;
24
25     node *z;
26     if (rand() % (x -> size + y -> size) < x -> size) {
27         z = copied(y);
28         z -> ch[0] = merge(x, y -> ch[0]);
29     }
30     else {
31         z = copied(x);
32         z -> ch[1] = merge(x -> ch[1], y);
33     }
34
35     z -> refresh(); // 因为每次只有一边会递归到儿子, 所
36     ↪ 以z不可能取到null
37     return z;
38 }
39
40 pair<node*, node*> split(node *x, int k) { // 左边大小为k
41     if (x == null)
42         return make_pair(null, null);
43
44     pair<node*, node*> pi(null, null);
45
46     if (k <= x -> ch[0] -> size) {
47         pi = split(x -> ch[0], k);
48
49         node *z = copied(x);
50         z -> ch[0] = pi.second;
51         z -> refresh();
52         pi.second = z;
53     }
54     else {
55         pi = split(x -> ch[1], k - x -> ch[0] -> size - 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 }
65
66 // 记得初始化null
67 int main() {
68     for (int i = 0; i <= n; i++)
69         null[i].ch[0] = null[i].ch[1] = null;
70     null -> size = 0;
71
72     // do something
73
74     return 0;
75 }

```

#### 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         if (dir(x) == dir(x -> p))
46             rot(x -> p -> p, dir(x -> p) ^ 1);
47         else
48             rot(x -> p, dir(x) ^ 1);
49         rot(x -> p, dir(x) ^ 1);
50     }
51 }
52
53 node *kth(int k, node *o) {
54     int d;
55     k++; // 因为最左边有一个哨兵
56
57     while (o != null) {
58         o -> pushdown();
59
60         if (k == o -> ch[0] -> size + 1)
61             return o;
62
63         if ((d = k > o -> ch[0] -> size))
64             k -= o -> ch[0] -> size + 1;
65         o = o -> ch[d];
66     }
67
68     return null;
69 }
70
71 void reverse(int l, int r) {

```

```

72     splay(kth(l - 1));
73     splay(kth(r + 1), root);
74
75     root -> ch[1] -> ch[0] -> rev ^= true;
76 }
77
78 int n, m;
79
80 int main() {
81     null -> size = 0;
82     null -> ch[0] = null -> ch[1] = null -> p = null;
83
84     scanf("%d%d", &n, &m);
85     root = null + n + 1;
86     root -> ch[0] = root -> ch[1] = root -> p = null;
87
88     for (int i = 1; i <= n; i++) {
89         null[i].ch[1] = null[i].p = null;
90         null[i].ch[0] = root;
91         root -> p = null + i;
92         (root = null + i) -> refresh();
93     }
94
95     null[n + 2].ch[1] = null[n + 2].p = null;
96     null[n + 2].ch[0] = root; // 这里直接建成一条链的, 如果
97     ⇨ 想减少常数也可以递归建一个平衡的树
98     root -> p = null + n + 2; // 总之记得建两个哨兵, 这
99     ⇨ 样splay起来不需要特判
100    (root = null + n + 2) -> refresh();
101
102    // Do something
103
104    return 0;
105 }

```

## 4.5 树链剖分

### 4.5.1 动态树形DP(最大权独立集)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 100005, maxm = 262155, inf =
6     ⇨ 0x3f3f3f3f;
7
8 struct binary_heap {
9     priority_queue<int> q, t;
10
11     binary_heap() {}
12
13     void push(int x) {
14         q.push(x);
15     }
16
17     void erase(int x) {
18         t.push(x);
19     }
20
21     int top() {
22         while (!t.empty() && q.top() == t.top()) {
23             q.pop();
24             t.pop();
25         }
26
27         return q.top();
28     }
29 } heap;

```

```

29 int pool[maxm][2][2], (*pt)[2][2] = pool;
30 void merge(int a[2][2], int b[2][2]) {
31     static int c[2][2];
32     memset(c, 0, sizeof(c));
33
34     for (int i = 0; i < 2; i++)
35         for (int j = 0; j < 2; j++)
36             for (int k = 0; k < 2; k++)
37                 if (!(j && k))
38                     for (int t = 0; t < 2; t++)
39                         c[i][t] = max(c[i][t], a[i][j] +
40                                         b[k][t]);
41
42     memcpy(a, c, sizeof(c));
43 }
44 vector<pair<int, int> > tw;
45 struct seg_tree {
46     int (*tr)[2][2], n;
47
48     int s, d[2];
49
50     seg_tree() {}
51
52     void update(int o) {
53         memcpy(tr[o], tr[o * 2], sizeof(int) * 4);
54         merge(tr[o], tr[o * 2 + 1]);
55     }
56
57     void build(int l, int r, int o) {
58         if (l == r) {
59             tr[o][0][0] = tw[l - 1].first;
60             tr[o][0][1] = tr[o][1][0] = -inf;
61             tr[o][1][1] = tw[l - 1].second;
62
63             return;
64         }
65
66         int mid = (l + r) / 2;
67
68         build(l, mid, o * 2);
69         build(mid + 1, r, o * 2 + 1);
70
71         update(o);
72     }
73
74     void modify(int l, int r, int o) {
75         if (l == r) {
76             tr[o][0][0] = d[0];
77             tr[o][0][1] = tr[o][1][0] = -inf;
78             tr[o][1][1] = d[1];
79
80             return;
81         }
82
83         int mid = (l + r) / 2;
84
85         if (s <= mid)
86             modify(l, mid, o * 2);
87         else
88             modify(mid + 1, r, o * 2 + 1);
89
90         update(o);
91     }
92
93     int getval() {
94         int ans = 0;

```

```

95         for (int i = 0; i < 2; i++)
96             for (int j = 0; j < 2; j++)
97                 ans = max(ans, tr[1][i][j]);
98
99         return ans;
100     }
101
102     pair<int, int> getpair() {
103         int ans[2] = {0};
104         for (int i = 0; i < 2; i++)
105             for (int j = 0; j < 2; j++)
106                 ans[i] = max(ans[i], tr[1][i][j]);
107
108         return make_pair(ans[0], ans[1]);
109     }
110
111     void build(int len) {
112         n = len;
113         int N = 1;
114         while (N < n * 2)
115             N *= 2;
116
117         tr = pt;
118         pt += N;
119
120         build(1, n, 1);
121     }
122
123     void modify(int x, int dat[2]) {
124         s = x;
125         for (int i = 0; i < 2; i++)
126             d[i] = dat[i];
127         modify(1, n, 1);
128     }
129 } seg[maxm];
130
131 vector<int> G[maxm];
132
133 int p[maxm], d[maxm], sz[maxm], son[maxm], top[maxm];
134 int dp[maxm][2], dptr[maxm][2], w[maxm];
135
136 void dfs1(int x) {
137     d[x] = d[p[x]] + 1;
138     sz[x] = 1;
139
140     for (int y : G[x])
141         if (y != p[x]) {
142             p[y] = x;
143             dfs1(y);
144
145             if (sz[y] > sz[son[x]])
146                 son[x] = y;
147
148             sz[x] += sz[y];
149         }
150     }
151
152 void dfs2(int x) {
153     if (x == son[p[x]])
154         top[x] = top[p[x]];
155     else
156         top[x] = x;
157
158     for (int y : G[x])
159         if (y != p[x])
160             dfs2(y);
161
162     dp[x][1] = w[x];
163     for (int y : G[x])

```

```

167     if (y != p[x] && y != son[x]) {
168         dp[x][1] += dptr[y][0];
169         dp[x][0] += max(dptr[y][0], dptr[y][1]);
170     }
171
172     if (top[x] == x) {
173         tw.clear();
174
175         for (int u = x; u; u = son[u])
176             tw.push_back(make_pair(dp[u][0], dp[u][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, dp[x]);
199         heap.push(seg[top[x]].getval());
200
201         tie(dptr[top[x]][0], dptr[top[x]][1]) =
202             ↪ seg[top[x]].getpair();
203
204         if (p[top[x]]) {
205             dp[p[top[x]]][0] += max(dptr[top[x]][0],
206                                     ↪ dptr[top[x]][1]);
207             dp[p[top[x]]][1] += dptr[top[x]][0];
208         }
209
210         x = p[top[x]];
211     }
212 }
213
214 int main() {
215     int n, m;
216     scanf("%d%d", &n, &m);
217
218     for (int i = 1; i <= n; i++)
219         scanf("%d", &w[i]);
220
221     for (int i = 1; i < n; i++) {
222         int x, y;
223         scanf("%d%d", &x, &y);
224
225         G[x].push_back(y);
226         G[y].push_back(x);
227     }
228
229     dfs1(1);
230     dfs2(1);
231
232     while (m--) {
233         int x, dat;
234         scanf("%d%d", &x, &dat);
235     }

```

```

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]; //
9   ↪ id是对应层所在子树的根
10 int a[maxn], ca[maxn], b[maxn][20], cb[maxn][20]; // 维护距
11   ↪ 离和用的
12 bool vis[maxn], col[maxn];
13
14 // 建树 总计O(n\log n)
15 // 需要调用找重心和预处理距离，同时递归调用自身
16 void build(int x, int k, int s, int pr) { // 结点，深度，连
17   ↪ 通块大小，点分树上的父亲
18     x = getcenter(x, s);
19     vis[x] = true;
20     depth[x] = k;
21     p[x] = pr;
22
23     for (int i = 0; i < (int)G[x].size(); i++)
24         if (!vis[G[x][i]]) {
25             d[G[x][i]][k] = W[x][i];
26             p[G[x][i]] = x;
27
28             getdis(G[x][i], k, G[x][i]); // bfs每个子树，预处
29   ↪ 理距离
30         }
31
32     for (int i = 0; i < (int)G[x].size(); i++)
33         if (!vis[G[x][i]])
34             build(G[x][i], k + 1, size[G[x][i]], x); // 递
35   ↪ 归建树
36 }
37
38 // 找重心 O(n)
39 int getcenter(int x, int s) {
40     int head = 0, tail = 0;
41     q[tail++] = x;
42
43     while (head != tail) {
44         x = q[head++];
45         size[x] = 1; // 这里不需要清空，因为以后要用的话一
46   ↪ 定会重新赋值
47         son[x] = 0;
48
49         for (int i = 0; i < (int)G[x].size(); i++)
50             if (!vis[G[x][i]] && G[x][i] != p[x]) {
51                 p[G[x][i]] = x;
52                 q[tail++] = G[x][i];
53             }
54     }
55
56     for (int i = tail - 1; i; i--) {
57         x = q[i];

```



```

51     size[p[x]] += size[x];
52
53     if (size[x] > size[son[p[x]]])
54         son[p[x]] = x;
55 }
56
57 x = q[0];
58 while (son[x] && size[son[x]] * 2 >= s)
59     x = son[x];
60
61 return x;
62 }
63
64 // 预处理距离  $O(n)$ 
65 // 方便起见, 这里直接用了笨一点的方法,  $O(n \log n)$  全存下来
66 void getdis(int x, int k, int rt) {
67     int head = 0, tail = 0;
68     q[tail++] = x;
69
70     while (head != tail) {
71         x = q[head++];
72         size[x] = 1;
73         id[x][k] = rt;
74
75         for (int i = 0; i < (int)G[x].size(); i++)
76             if (!vis[G[x][i]] && G[x][i] != p[x]) {
77                 p[G[x][i]] = x;
78                 d[G[x][i]][k] = d[x][k] + W[x][i];
79
80                 q[tail++] = G[x][i];
81             }
82     }
83
84     for (int i = tail - 1; i; i--)
85         size[p[q[i]]] += size[q[i]]; // 后面递归建树要用到
86                                     // 子问题大小
87 }
88
89 // 修改  $O(\log n)$ 
90 void modify(int x) {
91     if (col[x])
92         ca[x]--;
93     else
94         ca[x]++; // 记得先特判自己作为重心的那层
95
96     for (int u = p[x], k = depth[x] - 1; u; u = p[u], k--)
97         ↪ {
98             if (col[x]) {
99                 a[u] -= d[x][k];
100                 ca[u]--;
101
102                 b[id[x][k]][k] -= d[x][k];
103                 cb[id[x][k]][k]--;
104             }
105             else {
106                 a[u] += d[x][k];
107                 ca[u]++;
108
109                 b[id[x][k]][k] += d[x][k];
110                 cb[id[x][k]][k]++;
111             }
112         }
113
114     col[x] ^= true;
115 }
116
117 // 询问  $O(\log n)$ 
118 int query(int x) {
119     int ans = a[x]; // 特判自己是重心的那层

```

```

119     for (int u = p[x], k = depth[x] - 1; u; u = p[u], k--)
120         ans += a[u] - b[id[x][k]][k] + d[x][k] * (ca[u] -
121             ↪ cb[id[x][k]][k]);
122
123     return ans;
124 }

```

#### 4.6.2 紫荆花之恋

稍微重构了一下, 修改了代码风格.

另外这个是BFS版本, 跑得比DFS要快不少. (虽然主要复杂度并不在重构上)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 constexpr int maxn = 100005, maxk = 49;
6 constexpr double alpha = .75;
7
8 mt19937 rnd(23333333);
9
10 struct node {
11     int key, size, p;
12     node *ch[2];
13
14     node() {}
15
16     node(int key) : key(key), size(1), p(rnd()) {}
17
18     inline void update() {
19         size = ch[0] -> size + ch[1] -> size + 1;
20     }
21 } null[maxn * maxk], *pt = null;
22
23 vector<node*> pool;
24
25 node *newnode(int val) {
26     node *x;
27
28     if (!pool.empty()) {
29         x = pool.back();
30         pool.pop_back();
31     }
32     else
33         x = ++pt;
34
35     *x = node(val);
36     x -> ch[0] = x -> ch[1] = null;
37
38     return x;
39 }
40
41 void rot(node *&x, int d) {
42     node *y = x -> ch[d ^ 1];
43     x -> ch[d ^ 1] = y -> ch[d];
44     y -> ch[d] = x;
45
46     x -> update();
47     (x = y) -> update();
48 }
49
50 void insert(node *&o, int x) {
51     if (o == null) {
52         o = newnode(x);
53         return;
54     }
55
56     int d = (x > o -> key);

```

```

57     insert(o -> ch[d], x);
58     o -> update();
59
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
90     ↪ 好了
91     node *rt;
92
93     my_tree() : rt(null) {}
94
95     void clear() {
96         ::destroy(rt);
97         rt = null;
98     }
99
100     void insert(int x) {
101         ::insert(rt, x);
102     }
103
104     int order_of_key(int x) { // less than x
105         return ::get_order(rt, x);
106     }
107 } tr[maxn], tre[maxn][maxk];
108
109 vector<pair<int, int> > G[maxn];
110
111 int p[maxn], depth[maxn], d[maxn][maxk], rid[maxn][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++) {

```

```

126         tre[x][i].clear();
127         d[x][i] = rid[x][i] = siz[x][i] = 0;
128     }
129
130     for (auto pi : G[x]) {
131         int y = pi.first;
132
133         if (vis[y] && depth[y] >= depth[o]) {
134             vis[y] = false;
135             q[tail++] = y;
136         }
137     }
138 }
139
140
141 int getcenter(int o, int s) {
142     int head = 0, tail = 0;
143     q[tail++] = o;
144
145     while (head != tail) {
146         int x = q[head++];
147         sz[x] = 1;
148
149         for (auto pi : G[x]) {
150             int y = pi.first;
151
152             if (!vis[y] && y != p[x]) {
153                 p[y] = x;
154                 q[tail++] = y;
155             }
156         }
157     }
158
159     for (int i = s - 1; i; i--)
160         sz[p[q[i]]] += sz[q[i]];
161
162     int x = o;
163     while (true) {
164         bool ok = false;
165
166         for (auto pi : G[x]) {
167             int y = pi.first;
168             if (!vis[y] && y != p[x] && sz[y] * 2 > s) {
169                 x = y;
170                 ok = true;
171                 break;
172             }
173         }
174
175         if (!ok)
176             break;
177     }
178
179     return x;
180 }
181
182 void getdis(int st, int o, int k) {
183     int head = 0, tail = 0;
184     q[tail++] = st;
185
186     while (head != tail) {
187         int x = q[head++];
188         sz[x] = 1;
189         rid[x][k] = st;
190
191         tr[o].insert(d[x][k] - w[x]);
192         tre[st][k].insert(d[x][k] - w[x]);
193
194         for (auto pi : G[x]) {

```

```

195     int y = pi.first, val = pi.second;
196
197     if (!vis[y] && y != p[x]) {
198         p[y] = x;
199         d[y][k] = d[x][k] + val;
200         q[tail++] = y;
201     }
202 }
203
204
205 for (int i = tail - 1; i; i--)
206     sz[p[q[i]]] += sz[q[i]];
207
208 siz[st][k] = sz[st];
209 }
210
211 void rebuild(int x, int s, int pr) {
212     x = getcenter(x, s);
213     vis[x] = true;
214     p[x] = pr;
215     depth[x] = depth[pr] + 1;
216     sz[x] = s;
217
218     tr[x].insert(-w[x]);
219
220     for (auto pi : G[x]) {
221         int y = pi.first, val = pi.second;
222
223         if (!vis[y]) {
224             p[y] = x;
225             d[y][depth[x]] = val;
226             getdis(y, x, depth[x]);
227         }
228     }
229
230     for (auto pi : G[x]) {
231         int y = pi.first;
232
233         if (!vis[y])
234             rebuild(y, sz[y], x);
235     }
236 }
237
238 long long add_node(int x, int nw) { // nw是边权
239     depth[x] = depth[p[x]] + 1;
240     sz[x] = 1;
241     vis[x] = true;
242
243     tr[x].insert(-w[x]);
244
245     long long tmp = 0;
246     int goat = 0; // 替罪羊
247
248     for (int u = p[x], k = depth[x] - 1; u; u = p[u], k--)
249         ↪ {
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] - d[x]
255                 ↪ [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;

```

```

263     }
264
265     if (goat) {
266         destroy(goat);
267         rebuild(goat, sz[goat], p[goat]);
268     }
269
270     return tmp;
271 }
272
273 int main() {
274
275     null -> ch[0] = null -> ch[1] = null;
276     null -> size = 0;
277
278     int n;
279     scanf("%d", &n);
280
281     scanf("%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", &nw);
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) != (x)
   ↪ -> p -> ch[1]) // 判断是不是Splay的根
2 #define dir(x) ((x) == (x) -> p -> ch[1]) // 判断它是它父亲
   ↪ 的左 / 右儿子
3
4 struct node { // 结点类定义
5     int size; // Splay的子树大小
6     node *ch[2], *p;
7
8     node() : size(1) {}
9     void refresh() {
10         size = ch[0] -> size + ch[1] -> size + 1;
11     } // 附加信息维护
12 } null[maxn];
13
14 // 在主函数开头加上这句初始化
15 null -> size = 0;
16
17 // 初始化结点
18 void initialize(node *x) {

```

```

19     x -> ch[0] = x -> ch[1] = x -> p = null;
20 }
21
22 // Access 均摊 $O(\log n)$ 
23 // LCT核心操作, 把结点到根的路径打通, 顺便把与重儿子的连边
    ↪ 变成轻边
24 // 需要调用splay
25 node *access(node *x) {
26     node *y = null;
27
28     while (x != null) {
29         splay(x);
30
31         x -> ch[1] = y;
32         (y = x) -> refresh();
33
34         x = x -> p;
35     }
36
37     return y;
38 }
39
40 // Link 均摊 $O(\log n)$ 
41 // 把x的父亲设为y
42 // 要求x必须为所在树的根节点, 否则会导致后续各种莫名其妙的
    ↪ 问题
43 // 需要调用splay
44 void link(node *x, node *y) {
45     splay(x);
46     x -> p = y;
47 }
48
49 // Cut 均摊 $O(\log n)$ 
50 // 把x与其父亲的连边断开
51 // x可以是所在树的根节点, 这时此操作没有任何实质效果
52 // 需要调用access和splay
53 void cut(node *x) {
54     access(x);
55     splay(x);
56
57     x -> ch[0] -> p = null;
58     x -> ch[0] = null;
59
60     x -> refresh();
61 }
62
63 // Splay 均摊 $O(\log n)$ 
64 // 需要调用旋转
65 void splay(node *x) {
66     while (!isroot(x)) {
67         if (isroot(x -> p)) {
68             rot(x -> p, dir(x) ^ 1);
69             break;
70         }
71
72         if (dir(x) == dir(x -> p))
73             rot(x -> p -> p, dir(x -> p) ^ 1);
74         else
75             rot(x -> p, dir(x) ^ 1);
76         rot(x -> p, dir(x) ^ 1);
77     }
78 }
79
80 // 旋转(LCT版本)  $O(1)$ 
81 // 平衡树基本操作
82 // 要求对应儿子必须存在, 否则会导致后续各种莫名其妙的问
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 -> ch[0] !
    ↪ = (x) && (x) -> p -> ch[1] != (x)))
2  #define dir(x) ((x) == (x) -> p -> ch[1])
3
4  using namespace std;
5
6  const int maxn = 200005;
7
8  struct node{
9      int key, mx, pos;
10     bool rev;
11     node *ch[2], *p;
12
13     node(int key = 0): key(key), mx(key), pos(-1),
        ↪ rev(false) {}
14
15     void pushdown() {
16         if (!rev)
17             return;
18
19         ch[0] -> rev ^= true;
20         ch[1] -> rev ^= true;
21         swap(ch[0], ch[1]);
22
23         if (pos != -1)
24             pos ^= 1;
25
26         rev = false;
27     }
28
29     void refresh() {
30         mx = key;
31         pos = -1;
32         if (ch[0] -> mx > mx) {
33             mx = ch[0] -> mx;
34             pos = 0;
35         }
36         if (ch[1] -> mx > mx) {
37             mx = ch[1] -> mx;
38             pos = 1;
39         }
40     }
41 } null[maxn * 2];
42
43 void init(node *x, int k) {
44     x -> ch[0] = x -> ch[1] = x -> p = null;
45     x -> key = x -> mx = k;
46 }
47
48 void rot(node *x, int d) {
49     node *y = x -> ch[d ^ 1];
50     if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
51         y -> ch[d] -> p = x;
52
53     y -> p = x -> p;
54     if (!isroot(x))

```

```

55     x -> p -> ch[dir(x)] = y;
56
57     (y -> ch[d] = x) -> p = y;
58
59     x -> refresh();
60     y -> refresh();
61 }
62
63 void splay(node *x) {
64     x -> pushdown();
65
66     while (!isroot(x)) {
67         if (isroot(x -> p))
68             x -> p -> p -> pushdown();
69         x -> p -> pushdown();
70         x -> pushdown();
71
72         if (isroot(x -> p)) {
73             rot(x -> p, dir(x) ^ 1);
74             break;
75         }
76
77         if (dir(x) == dir(x -> p))
78             rot(x -> p -> p, dir(x -> p) ^ 1);
79         else
80             rot(x -> p, dir(x) ^ 1);
81
82         rot(x -> p, dir(x) ^ 1);
83     }
84 }
85
86 node *access(node *x) {
87     node *y = null;
88
89     while (x != null) {
90         splay(x);
91
92         x -> ch[1] = y;
93         (y = x) -> refresh();
94
95         x = x -> p;
96     }
97
98     return y;
99 }
100
101 void makeroot(node *x) {
102     access(x);
103     splay(x);
104     x -> rev ^= true;
105 }
106
107 void link(node *x, node *y) {
108     makeroot(x);
109     x -> p = y;
110 }
111
112 void cut(node *x, node *y) {
113     makeroot(x);
114     access(y);
115     splay(y);
116
117     y -> ch[0] -> p = null;
118     y -> ch[0] = null;
119     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     makeroot(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]) - null -
154             ↪ 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，
  ↪ 没必要的时候就不要用
2 // 如果维护子树最小值就需要套一个可删除的堆来维护，复杂度
  ↪ 会变成 $O(n \log^2 n)$ 
3 // 注意由于这道题与边权有关，需要边权拆点变点权
4
5 // 宏定义
6 #define isroot(x) ((x) -> p == null || ((x) != (x) -> p ->
  ↪ ch[0] && (x) != (x) -> p -> ch[1]))
7 #define dir(x) ((x) == (x) -> p -> ch[1])
8
9 // 节点类定义
10 struct node { // 以维护子树中黑点到根距离和为列
11     int w, chain_cnt, tree_cnt;
12     long long sum, suml, sumr, tree_sum; // 由于换根需要子
  ↪ 树反转，需要维护两个方向的信息
13     bool rev, col;
14     node *ch[2], *p;
15
16     node() : w(0), chain_cnt(0),
  ↪ tree_cnt(0), sum(0), suml(0), sumr(0),
  ↪ tree_sum(0), rev(false), col(false) {}
17
18     inline void pushdown() {

```

```

20     if(!rev)
21         return;
22
23     ch[0]->rev ^= true;
24     ch[1]->rev ^= true;
25     swap(ch[0], ch[1]);
26     swap(suml, sumr);
27
28     rev = false;
29 }
30
31 inline void refresh() { // 如果不想这样特判就pushdown—
32     ↪ 下
33     // pushdown();
34
35     sum = ch[0] -> sum + ch[1] -> sum + w;
36     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
37         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
38         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
39         ↪ (ch[0] -> sum + w) + tree_sum;
40
41     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
42         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
43         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
44         ↪ (ch[1] -> sum + w) + tree_sum;
45
46     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
47         ↪ + tree_cnt;
48 }
49 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
50
51 // 封装构造函数
52 node *newnode(int w) {
53     node *x = nodes.front(); // 因为有删边加边, 可以用一个
54     ↪ 队列维护可用结点
55     nodes.pop();
56     initialize(x);
57     x -> w = w;
58     x -> refresh();
59     return x;
60 }
61
62 // 封装初始化函数
63 // 记得在进行操作之前对所有结点调用一遍
64 inline void initialize(node *x) {
65     *x = node();
66     x -> ch[0] = x -> ch[1] = x -> p = null;
67 }
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 -> ch[1] ->
79             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
80         x -> ch[1] = y;
81
82         (y = x) -> refresh();
83         x = x -> p;
84     }
85
86     return y;
87 }
88
89 // 找到一个点所在连通块的根
90 // 对比原版没有变化
91 node *getroot(node *x) {
92     if(!rev)
93         return;
94
95     ch[0]->rev ^= true;
96     ch[1]->rev ^= true;
97     swap(ch[0], ch[1]);
98     swap(suml, sumr);
99
100     rev = false;
101 }
102
103 inline void refresh() { // 如果不想这样特判就pushdown—
104     ↪ 下
105     // pushdown();
106
107     sum = ch[0] -> sum + ch[1] -> sum + w;
108     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
109         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
110         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
111         ↪ (ch[0] -> sum + w) + tree_sum;
112
113     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
114         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
115         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
116         ↪ (ch[1] -> sum + w) + tree_sum;
117
118     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
119         ↪ + tree_cnt;
120 }
121 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
122
123 // 封装构造函数
124 node *newnode(int w) {
125     node *x = nodes.front(); // 因为有删边加边, 可以用一个
126     ↪ 队列维护可用结点
127     nodes.pop();
128     initialize(x);
129     x -> w = w;
130     x -> refresh();
131     return x;
132 }
133
134 // 封装初始化函数
135 // 记得在进行操作之前对所有结点调用一遍
136 inline void initialize(node *x) {
137     *x = node();
138     x -> ch[0] = x -> ch[1] = x -> p = null;
139 }
140
141 // 注意一下在Access的同时更新子树信息的方法
142 node *access(node *x) {
143     node *y = null;
144
145     while (x != null) {
146         splay(x);
147
148         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
149             ↪ chain_cnt;
150         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
151             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
152         x -> ch[1] = y;
153
154         (y = x) -> refresh();
155         x = x -> p;
156     }
157
158     return y;
159 }
160
161 // 找到一个点所在连通块的根
162 // 对比原版没有变化
163 node *getroot(node *x) {
164     if(!rev)
165         return;
166
167     ch[0]->rev ^= true;
168     ch[1]->rev ^= true;
169     swap(ch[0], ch[1]);
170     swap(suml, sumr);
171
172     rev = false;
173 }
174
175 inline void refresh() { // 如果不想这样特判就pushdown—
176     ↪ 下
177     // pushdown();
178
179     sum = ch[0] -> sum + ch[1] -> sum + w;
180     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
181         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
182         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
183         ↪ (ch[0] -> sum + w) + tree_sum;
184
185     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
186         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
187         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
188         ↪ (ch[1] -> sum + w) + tree_sum;
189
190     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
191         ↪ + tree_cnt;
192 }
193 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
194
195 // 封装构造函数
196 node *newnode(int w) {
197     node *x = nodes.front(); // 因为有删边加边, 可以用一个
198     ↪ 队列维护可用结点
199     nodes.pop();
200     initialize(x);
201     x -> w = w;
202     x -> refresh();
203     return x;
204 }
205
206 // 封装初始化函数
207 // 记得在进行操作之前对所有结点调用一遍
208 inline void initialize(node *x) {
209     *x = node();
210     x -> ch[0] = x -> ch[1] = x -> p = null;
211 }
212
213 // 注意一下在Access的同时更新子树信息的方法
214 node *access(node *x) {
215     node *y = null;
216
217     while (x != null) {
218         splay(x);
219
220         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
221             ↪ chain_cnt;
222         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
223             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
224         x -> ch[1] = y;
225
226         (y = x) -> refresh();
227         x = x -> p;
228     }
229
230     return y;
231 }
232
233 // 找到一个点所在连通块的根
234 // 对比原版没有变化
235 node *getroot(node *x) {
236     if(!rev)
237         return;
238
239     ch[0]->rev ^= true;
240     ch[1]->rev ^= true;
241     swap(ch[0], ch[1]);
242     swap(suml, sumr);
243
244     rev = false;
245 }
246
247 inline void refresh() { // 如果不想这样特判就pushdown—
248     ↪ 下
249     // pushdown();
250
251     sum = ch[0] -> sum + ch[1] -> sum + w;
252     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
253         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
254         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
255         ↪ (ch[0] -> sum + w) + tree_sum;
256
257     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
258         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
259         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
260         ↪ (ch[1] -> sum + w) + tree_sum;
261
262     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
263         ↪ + tree_cnt;
264 }
265 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
266
267 // 封装构造函数
268 node *newnode(int w) {
269     node *x = nodes.front(); // 因为有删边加边, 可以用一个
270     ↪ 队列维护可用结点
271     nodes.pop();
272     initialize(x);
273     x -> w = w;
274     x -> refresh();
275     return x;
276 }
277
278 // 封装初始化函数
279 // 记得在进行操作之前对所有结点调用一遍
280 inline void initialize(node *x) {
281     *x = node();
282     x -> ch[0] = x -> ch[1] = x -> p = null;
283 }
284
285 // 注意一下在Access的同时更新子树信息的方法
286 node *access(node *x) {
287     node *y = null;
288
289     while (x != null) {
290         splay(x);
291
292         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
293             ↪ chain_cnt;
294         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
295             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
296         x -> ch[1] = y;
297
298         (y = x) -> refresh();
299         x = x -> p;
300     }
301
302     return y;
303 }
304
305 // 找到一个点所在连通块的根
306 // 对比原版没有变化
307 node *getroot(node *x) {
308     if(!rev)
309         return;
310
311     ch[0]->rev ^= true;
312     ch[1]->rev ^= true;
313     swap(ch[0], ch[1]);
314     swap(suml, sumr);
315
316     rev = false;
317 }
318
319 inline void refresh() { // 如果不想这样特判就pushdown—
320     ↪ 下
321     // pushdown();
322
323     sum = ch[0] -> sum + ch[1] -> sum + w;
324     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
325         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
326         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
327         ↪ (ch[0] -> sum + w) + tree_sum;
328
329     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
330         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
331         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
332         ↪ (ch[1] -> sum + w) + tree_sum;
333
334     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
335         ↪ + tree_cnt;
336 }
337 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
338
339 // 封装构造函数
340 node *newnode(int w) {
341     node *x = nodes.front(); // 因为有删边加边, 可以用一个
342     ↪ 队列维护可用结点
343     nodes.pop();
344     initialize(x);
345     x -> w = w;
346     x -> refresh();
347     return x;
348 }
349
350 // 封装初始化函数
351 // 记得在进行操作之前对所有结点调用一遍
352 inline void initialize(node *x) {
353     *x = node();
354     x -> ch[0] = x -> ch[1] = x -> p = null;
355 }
356
357 // 注意一下在Access的同时更新子树信息的方法
358 node *access(node *x) {
359     node *y = null;
360
361     while (x != null) {
362         splay(x);
363
364         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
365             ↪ chain_cnt;
366         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
367             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
368         x -> ch[1] = y;
369
370         (y = x) -> refresh();
371         x = x -> p;
372     }
373
374     return y;
375 }
376
377 // 找到一个点所在连通块的根
378 // 对比原版没有变化
379 node *getroot(node *x) {
380     if(!rev)
381         return;
382
383     ch[0]->rev ^= true;
384     ch[1]->rev ^= true;
385     swap(ch[0], ch[1]);
386     swap(suml, sumr);
387
388     rev = false;
389 }
390
391 inline void refresh() { // 如果不想这样特判就pushdown—
392     ↪ 下
393     // pushdown();
394
395     sum = ch[0] -> sum + ch[1] -> sum + w;
396     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
397         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
398         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
399         ↪ (ch[0] -> sum + w) + tree_sum;
400
401     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
402         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
403         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
404         ↪ (ch[1] -> sum + w) + tree_sum;
405
406     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
407         ↪ + tree_cnt;
408 }
409 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
410
411 // 封装构造函数
412 node *newnode(int w) {
413     node *x = nodes.front(); // 因为有删边加边, 可以用一个
414     ↪ 队列维护可用结点
415     nodes.pop();
416     initialize(x);
417     x -> w = w;
418     x -> refresh();
419     return x;
420 }
421
422 // 封装初始化函数
423 // 记得在进行操作之前对所有结点调用一遍
424 inline void initialize(node *x) {
425     *x = node();
426     x -> ch[0] = x -> ch[1] = x -> p = null;
427 }
428
429 // 注意一下在Access的同时更新子树信息的方法
430 node *access(node *x) {
431     node *y = null;
432
433     while (x != null) {
434         splay(x);
435
436         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
437             ↪ chain_cnt;
438         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
439             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
440         x -> ch[1] = y;
441
442         (y = x) -> refresh();
443         x = x -> p;
444     }
445
446     return y;
447 }
448
449 // 找到一个点所在连通块的根
450 // 对比原版没有变化
451 node *getroot(node *x) {
452     if(!rev)
453         return;
454
455     ch[0]->rev ^= true;
456     ch[1]->rev ^= true;
457     swap(ch[0], ch[1]);
458     swap(suml, sumr);
459
460     rev = false;
461 }
462
463 inline void refresh() { // 如果不想这样特判就pushdown—
464     ↪ 下
465     // pushdown();
466
467     sum = ch[0] -> sum + ch[1] -> sum + w;
468     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
469         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
470         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
471         ↪ (ch[0] -> sum + w) + tree_sum;
472
473     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
474         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
475         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
476         ↪ (ch[1] -> sum + w) + tree_sum;
477
478     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
479         ↪ + tree_cnt;
480 }
481 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
482
483 // 封装构造函数
484 node *newnode(int w) {
485     node *x = nodes.front(); // 因为有删边加边, 可以用一个
486     ↪ 队列维护可用结点
487     nodes.pop();
488     initialize(x);
489     x -> w = w;
490     x -> refresh();
491     return x;
492 }
493
494 // 封装初始化函数
495 // 记得在进行操作之前对所有结点调用一遍
496 inline void initialize(node *x) {
497     *x = node();
498     x -> ch[0] = x -> ch[1] = x -> p = null;
499 }
500
501 // 注意一下在Access的同时更新子树信息的方法
502 node *access(node *x) {
503     node *y = null;
504
505     while (x != null) {
506         splay(x);
507
508         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
509             ↪ chain_cnt;
510         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
511             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
512         x -> ch[1] = y;
513
514         (y = x) -> refresh();
515         x = x -> p;
516     }
517
518     return y;
519 }
520
521 // 找到一个点所在连通块的根
522 // 对比原版没有变化
523 node *getroot(node *x) {
524     if(!rev)
525         return;
526
527     ch[0]->rev ^= true;
528     ch[1]->rev ^= true;
529     swap(ch[0], ch[1]);
530     swap(suml, sumr);
531
532     rev = false;
533 }
534
535 inline void refresh() { // 如果不想这样特判就pushdown—
536     ↪ 下
537     // pushdown();
538
539     sum = ch[0] -> sum + ch[1] -> sum + w;
540     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
541         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
542         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
543         ↪ (ch[0] -> sum + w) + tree_sum;
544
545     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
546         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
547         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
548         ↪ (ch[1] -> sum + w) + tree_sum;
549
550     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
551         ↪ + tree_cnt;
552 }
553 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
554
555 // 封装构造函数
556 node *newnode(int w) {
557     node *x = nodes.front(); // 因为有删边加边, 可以用一个
558     ↪ 队列维护可用结点
559     nodes.pop();
560     initialize(x);
561     x -> w = w;
562     x -> refresh();
563     return x;
564 }
565
566 // 封装初始化函数
567 // 记得在进行操作之前对所有结点调用一遍
568 inline void initialize(node *x) {
569     *x = node();
570     x -> ch[0] = x -> ch[1] = x -> p = null;
571 }
572
573 // 注意一下在Access的同时更新子树信息的方法
574 node *access(node *x) {
575     node *y = null;
576
577     while (x != null) {
578         splay(x);
579
580         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
581             ↪ chain_cnt;
582         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
583             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
584         x -> ch[1] = y;
585
586         (y = x) -> refresh();
587         x = x -> p;
588     }
589
590     return y;
591 }
592
593 // 找到一个点所在连通块的根
594 // 对比原版没有变化
595 node *getroot(node *x) {
596     if(!rev)
597         return;
598
599     ch[0]->rev ^= true;
600     ch[1]->rev ^= true;
601     swap(ch[0], ch[1]);
602     swap(suml, sumr);
603
604     rev = false;
605 }
606
607 inline void refresh() { // 如果不想这样特判就pushdown—
608     ↪ 下
609     // pushdown();
610
611     sum = ch[0] -> sum + ch[1] -> sum + w;
612     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
613         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
614         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
615         ↪ (ch[0] -> sum + w) + tree_sum;
616
617     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
618         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
619         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
620         ↪ (ch[1] -> sum + w) + tree_sum;
621
622     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
623         ↪ + tree_cnt;
624 }
625 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
626
627 // 封装构造函数
628 node *newnode(int w) {
629     node *x = nodes.front(); // 因为有删边加边, 可以用一个
630     ↪ 队列维护可用结点
631     nodes.pop();
632     initialize(x);
633     x -> w = w;
634     x -> refresh();
635     return x;
636 }
637
638 // 封装初始化函数
639 // 记得在进行操作之前对所有结点调用一遍
640 inline void initialize(node *x) {
641     *x = node();
642     x -> ch[0] = x -> ch[1] = x -> p = null;
643 }
644
645 // 注意一下在Access的同时更新子树信息的方法
646 node *access(node *x) {
647     node *y = null;
648
649     while (x != null) {
650         splay(x);
651
652         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
653             ↪ chain_cnt;
654         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
655             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
656         x -> ch[1] = y;
657
658         (y = x) -> refresh();
659         x = x -> p;
660     }
661
662     return y;
663 }
664
665 // 找到一个点所在连通块的根
666 // 对比原版没有变化
667 node *getroot(node *x) {
668     if(!rev)
669         return;
670
671     ch[0]->rev ^= true;
672     ch[1]->rev ^= true;
673     swap(ch[0], ch[1]);
674     swap(suml, sumr);
675
676     rev = false;
677 }
678
679 inline void refresh() { // 如果不想这样特判就pushdown—
680     ↪ 下
681     // pushdown();
682
683     sum = ch[0] -> sum + ch[1] -> sum + w;
684     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
685         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
686         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
687         ↪ (ch[0] -> sum + w) + tree_sum;
688
689     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
690         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
691         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
692         ↪ (ch[1] -> sum + w) + tree_sum;
693
694     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
695         ↪ + tree_cnt;
696 }
697 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
698
699 // 封装构造函数
700 node *newnode(int w) {
701     node *x = nodes.front(); // 因为有删边加边, 可以用一个
702     ↪ 队列维护可用结点
703     nodes.pop();
704     initialize(x);
705     x -> w = w;
706     x -> refresh();
707     return x;
708 }
709
710 // 封装初始化函数
711 // 记得在进行操作之前对所有结点调用一遍
712 inline void initialize(node *x) {
713     *x = node();
714     x -> ch[0] = x -> ch[1] = x -> p = null;
715 }
716
717 // 注意一下在Access的同时更新子树信息的方法
718 node *access(node *x) {
719     node *y = null;
720
721     while (x != null) {
722         splay(x);
723
724         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
725             ↪ chain_cnt;
726         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
727             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
728         x -> ch[1] = y;
729
730         (y = x) -> refresh();
731         x = x -> p;
732     }
733
734     return y;
735 }
736
737 // 找到一个点所在连通块的根
738 // 对比原版没有变化
739 node *getroot(node *x) {
740     if(!rev)
741         return;
742
743     ch[0]->rev ^= true;
744     ch[1]->rev ^= true;
745     swap(ch[0], ch[1]);
746     swap(suml, sumr);
747
748     rev = false;
749 }
750
751 inline void refresh() { // 如果不想这样特判就pushdown—
752     ↪ 下
753     // pushdown();
754
755     sum = ch[0] -> sum + ch[1] -> sum + w;
756     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
757         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
758         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
759         ↪ (ch[0] -> sum + w) + tree_sum;
760
761     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
762         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
763         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
764         ↪ (ch[1] -> sum + w) + tree_sum;
765
766     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
767         ↪ + tree_cnt;
768 }
769 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
770
771 // 封装构造函数
772 node *newnode(int w) {
773     node *x = nodes.front(); // 因为有删边加边, 可以用一个
774     ↪ 队列维护可用结点
775     nodes.pop();
776     initialize(x);
777     x -> w = w;
778     x -> refresh();
779     return x;
780 }
781
782 // 封装初始化函数
783 // 记得在进行操作之前对所有结点调用一遍
784 inline void initialize(node *x) {
785     *x = node();
786     x -> ch[0] = x -> ch[1] = x -> p = null;
787 }
788
789 // 注意一下在Access的同时更新子树信息的方法
790 node *access(node *x) {
791     node *y = null;
792
793     while (x != null) {
794         splay(x);
795
796         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
797             ↪ chain_cnt;
798         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
799             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
800         x -> ch[1] = y;
801
802         (y = x) -> refresh();
803         x = x -> p;
804     }
805
806     return y;
807 }
808
809 // 找到一个点所在连通块的根
810 // 对比原版没有变化
811 node *getroot(node *x) {
812     if(!rev)
813         return;
814
815     ch[0]->rev ^= true;
816     ch[1]->rev ^= true;
817     swap(ch[0], ch[1]);
818     swap(suml, sumr);
819
820     rev = false;
821 }
822
823 inline void refresh() { // 如果不想这样特判就pushdown—
824     ↪ 下
825     // pushdown();
826
827     sum = ch[0] -> sum + ch[1] -> sum + w;
828     suml = (ch[0] -> rev ? ch[0] -> sumr : ch[0] ->
829         ↪ suml) + (ch[1] -> rev ? ch[1] -> sumr : ch[1]
830         ↪ -> suml) + (tree_cnt + ch[1] -> chain_cnt) *
831         ↪ (ch[0] -> sum + w) + tree_sum;
832
833     sumr = (ch[0] -> rev ? ch[0] -> suml : ch[0] ->
834         ↪ sumr) + (ch[1] -> rev ? ch[1] -> suml : ch[1]
835         ↪ -> sumr) + (tree_cnt + ch[0] -> chain_cnt) *
836         ↪ (ch[1] -> sum + w) + tree_sum;
837
838     chain_cnt = ch[0] -> chain_cnt + ch[1] -> chain_cnt
839         ↪ + tree_cnt;
840 }
841 } null[maxn * 2]; // 如果没有边权变点权就不用乘2了
842
843 // 封装构造函数
844 node *newnode(int w) {
845     node *x = nodes.front(); // 因为有删边加边, 可以用一个
846     ↪ 队列维护可用结点
847     nodes.pop();
848     initialize(x);
849     x -> w = w;
850     x -> refresh();
851     return x;
852 }
853
854 // 封装初始化函数
855 // 记得在进行操作之前对所有结点调用一遍
856 inline void initialize(node *x) {
857     *x = node();
858     x -> ch[0] = x -> ch[1] = x -> p = null;
859 }
860
861 // 注意一下在Access的同时更新子树信息的方法
862 node *access(node *x) {
863     node *y = null;
864
865     while (x != null) {
866         splay(x);
867
868         x -> tree_cnt += x -> ch[1] -> chain_cnt - y ->
869             ↪ chain_cnt;
870         x -> tree_sum += (x -> ch[1] -> rev ? x -> ch[1] ->
871             ↪ sumr : x -> ch[1] -> suml) - y -> suml;
872         x -> ch[1] = y;
873
874         (y = x) -> refresh();
875         x = x -> p;
876     }
877
878     return y;
879 }
880
881 // 找到一个点所在连通块的根
882 // 对比原版没有变化
883 node *getroot(node *x) {
884     if(!rev)
885         return;
886
887     ch[0]->rev ^= true;
888     ch[1]->rev ^= true;
889     swap(ch[0], ch[1]);
890     swap(suml, sumr);
891
892     rev = false;
893 }
894
895 inline void refresh() { // 如果不想这样特判就pushdown—
896     ↪ 下
897     // pushdown();
898
899     sum = ch[0] -&
```



```

149     }
150 }
151
152 // 旋转函数
153 // 对比原版没有变化
154 void rot(node *x, int d) {
155     node *y = x -> ch[d ^ 1];
156
157     if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
158         y -> ch[d] -> p = x;
159
160     y -> p = x -> p;
161     if (!isroot(x))
162         x -> p -> ch[dir(x)] = y;
163
164     (y -> ch[d] = x) -> p = y;
165
166     x -> refresh();
167     y -> refresh();
168 }

```

#### 4.7.4 模板题:动态QTREE4(询问树上相距最远点)

```

1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 #include <ext/pb_ds/priority_queue.hpp>
5
6 #define isroot(x) ((x) -> p == null || ((x) != (x) -> p ->
    ↳ 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 const int maxn = 100010;
13 const long long INF = 1000000000000000000ll;
14
15 struct binary_heap {
16     __gnu_pbds::priority_queue<long long, less<long long>,
    ↳ binary_heap_tag> q1, q2;
17     binary_heap() {}
18
19     void push(long long x) {
20         if (x > (-INF) >> 2)
21             q1.push(x);
22     }
23
24     void erase(long long x) {
25         if (x > (-INF) >> 2)
26             q2.push(x);
27     }
28
29     long long top() {
30         if (empty())
31             return -INF;
32
33         while (!q2.empty() && q1.top() == q2.top()) {
34             q1.pop();
35             q2.pop();
36         }
37
38         return q1.top();
39     }
40
41     long long top2() {
42         if (size() < 2)
43             return -INF;
44
45         long long a = top();
46         erase(a);

```

```

47         long long b = top();
48         push(a);
49         return a + b;
50     }
51
52     int size() {
53         return q1.size() - q2.size();
54     }
55
56     bool empty() {
57         return q1.size() == q2.size();
58     }
59 } heap; // 全局堆维护每条链的最大子段和
60
61 struct node {
62     long long sum, maxsum, prefix, suffix;
63     int key;
64     binary_heap heap; // 每个点的堆存的是它的子树中到它的最
    ↳ 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), prefix(-INF),
        suffix(-INF), key(k), rev(false) {}
68     inline void pushdown() {
69         if (!rev)
70             return;
71
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] -> prefix);
86         suffix = max(ch[1] -> suffix,
87             ch[1] -> sum + key + ch[0] -> suffix);
88         maxsum = max(max(ch[0] -> maxsum, ch[1] -> maxsum),
89             ch[0] -> suffix + key + ch[1] ->
    ↳ prefix);
90
91         if (!heap.empty()) {
92             prefix = max(prefix,
93                 ch[0] -> sum + key + heap.top());
94             suffix = max(suffix,
95                 ch[1] -> sum + key + heap.top());
96             maxsum = max(maxsum, max(ch[0] -> suffix,
97                 ch[1] -> prefix) + key
    ↳ + heap.top());
98
99             if (heap.size() > 1) {
100                 maxsum = max(maxsum, heap.top2() + key);
101             }
102         }
103     }
104 } null[maxn << 1], *ptr = null;
105
106 void addedge(int, int, int);
107 void deletedge(int, int);
108 void modify(int, int, int);
109 void modify_color(int);
110 node *newnode(int);
111 node *access(node *);
112 void makeroot(node *);
113 void link(node *, node *);
114 void cut(node *, node *);
115 void splay(node *);

```

```

116 void rot(node *, int);
117
118 queue<node *>freenodes;
119 tree<pair<int, int>, node *>mp;
120
121 bool col[maxn] = {false};
122 char c;
123 int n, m, k, x, y, z;
124
125 int main() {
126     null -> ch[0] = null -> ch[1] = null -> p = null;
127     scanf("%d%d%d", &n, &m, &k);
128
129     for (int i = 1; i <= n; i++)
130         newnode(0);
131
132     heap.push(0);
133
134     while (k--) {
135         scanf("%d", &x);
136
137         col[x] = true;
138         null[x].heap.push(0);
139     }
140
141     for (int i = 1; i < n; i++) {
142         scanf("%d%d%d", &x, &y, &z);
143
144         if (x > y)
145             swap(x, y);
146         addedge(x, y, z);
147     }
148
149     while (m--) {
150         scanf(" %c%d", &c, &x);
151
152         if (c == 'A') {
153             scanf("%d", &y);
154
155             if (x > y)
156                 swap(x, y);
157             delede(x, y);
158         }
159         else if (c == 'B') {
160             scanf("%d%d", &y, &z);
161
162             if (x > y)
163                 swap(x, y);
164             addedge(x, y, z);
165         }
166         else if (c == 'C') {
167             scanf("%d%d", &y, &z);
168
169             if (x > y)
170                 swap(x, y);
171             modify(x, y, z);
172         }
173         else
174             modify_color(x);
175
176         printf("%lld\n", (heap.top() > 0 ? heap.top() :
177             ↪ -1));
178     }
179     return 0;
180 }
181
182 void addedge(int x, int y, int z) {
183     node *tmp;
184     if (freenodes.empty())
185         tmp = newnode(z);
186     else {

```

```

187         tmp = freenodes.front();
188         freenodes.pop();
189         *tmp = node(z);
190     }
191
192     tmp -> ch[0] = tmp -> ch[1] = tmp -> p = null;
193
194     heap.push(tmp -> maxsum);
195     link(tmp, null + x);
196     link(tmp, null + y);
197     mp[make_pair(x, y)] = tmp;
198 }
199
200 void delede(int x, int y) {
201     node *tmp = mp[make_pair(x, y)];
202
203     cut(tmp, null + x);
204     cut(tmp, null + y);
205
206     freenodes.push(tmp);
207     heap.erase(tmp -> maxsum);
208     mp.erase(make_pair(x, y));
209 }
210
211 void modify(int x, int y, int z) {
212     node *tmp = mp[make_pair(x, y)];
213     makeroot(tmp);
214     tmp -> pushdown();
215
216     heap.erase(tmp -> maxsum);
217     tmp -> key = z;
218     tmp -> refresh();
219     heap.push(tmp -> maxsum);
220 }
221
222 void modify_color(int x) {
223     makeroot(null + x);
224     col[x] ^= true;
225
226     if (col[x])
227         null[x].heap.push(0);
228     else
229         null[x].heap.erase(0);
230
231     heap.erase(null[x].maxsum);
232     null[x].refresh();
233     heap.push(null[x].maxsum);
234 }
235
236 node *newnode(int k) {
237     *(++ptr) = node(k);
238     ptr -> ch[0] = ptr -> ch[1] = ptr -> p = null;
239     return ptr;
240 }
241
242 node *access(node *x) {
243     splay(x);
244     heap.erase(x -> maxsum);
245     x -> refresh();
246
247     if (x -> ch[1] != null) {
248         x -> ch[1] -> pushdown();
249         x -> heap.push(x -> ch[1] -> prefix);
250         x -> refresh();
251         heap.push(x -> ch[1] -> maxsum);
252     }
253
254     x -> ch[1] = null;
255     x -> refresh();
256     node *y = x;
257     x = x -> p;
258 }

```

```

259 while (x != null) {
260     splay(x);
261     heap.erase(x -> maxsum);
262
263     if (x -> ch[1] != null) {
264         x -> ch[1] -> pushdown();
265         x -> heap.push(x -> ch[1] -> prefix);
266         heap.push(x -> ch[1] -> maxsum);
267     }
268
269     x -> heap.erase(y -> prefix);
270     x -> ch[1] = y;
271     (y = x) -> refresh();
272     x = x -> p;
273 }
274
275 heap.push(y -> maxsum);
276 return y;
277 }
278
279 void makeroot(node *x) {
280     access(x);
281     splay(x);
282     x -> rev ^= true;
283 }
284
285 void link(node *x, node *y) { // 新添一条虚边, 维护y对应的
    ↳ 堆
286     makeroot(x);
287     makeroot(y);
288
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, 一条链变成两
    ↳ 条链, 需要维护全局堆
298     makeroot(x);
299     access(y);
300     splay(y);
301
302     heap.erase(y -> maxsum);
303     heap.push(y -> ch[0] -> maxsum);
304     y -> ch[0] -> p = null;
305     y -> ch[0] = null;
306     y -> refresh();
307     heap.push(y -> maxsum);
308 }
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] ->
            ↳ l[0]));
16         l[1] = min(x[1], min(ch[0] -> l[1], ch[1] ->
            ↳ l[1]));
17         r[0] = max(x[0], max(ch[0] -> r[0], ch[1] ->
            ↳ r[0]));
18         r[1] = max(x[1], max(ch[0] -> r[1], ch[1] ->
            ↳ r[1]));
19     }
20 } null[maxn], *root = null;
21
22 void build(int l, int r, int k, node *&rt) {
23     if (l > r) {
24         rt = null;
25         return;
26     }
27
28     int mid = (l + r) / 2;
29
30     d = k;
31     nth_element(null + l, null + mid, null + r + 1);
32
33     rt = null + mid;
34     build(l, mid - 1, k ^ 1, rt -> ch[0]);
35     build(mid + 1, r, k ^ 1, rt -> ch[1]);
36
37     rt -> refresh();
38 }
39
40 void query(node *rt) {

```

```

41  if (l[0] <= rt -> l[0] && l[1] <= rt -> l[1] && rt ->
    ↳ r[0] <= r[0] && rt -> r[1] <= r[1]) {
42      ans += rt -> sum;
43      return;
44  }
45  else if (l[0] > rt -> r[0] || l[1] > rt -> r[1] || r[0]
    ↳ < rt -> l[0] || r[1] < rt -> l[1])
46      return;
47
48  if (l[0] <= rt -> x[0] && l[1] <= rt -> x[1] && rt ->
    ↳ x[0] <= r[0] && rt -> x[1] <= r[1])
49      ans += rt -> w;
50
51  query(rt -> ch[0]);
52  query(rt -> ch[1]);
53  }
54
55  int main() {
56
57      null -> l[0] = null -> l[1] = 10000000;
58      null -> r[0] = null -> r[1] = -10000000;
59      null -> sum = 0;
60      null -> ch[0] = null -> ch[1] = null;
61      scanf("%d");
62
63      while (scanf("%d", &op) == 1 && op != 3) {
64          if (op == 1) {
65              tmp++;
66              scanf("%d%d", &x[tmp][0], &x[tmp][1],
    ↳ &w[tmp]);
67              x[tmp][0] ^= ans;
68              x[tmp][1] ^= ans;
69              w[tmp] ^= ans;
70
71              if (tmp == B) {
72                  for (int i = 1; i <= tmp; i++) {
73                      null[cnt + i].x[0] = x[i][0];
74                      null[cnt + i].x[1] = x[i][1];
75                      null[cnt + i].w = w[i];
76                  }
77
78                  build(1, cnt += tmp, 0, root);
79                  tmp = 0;
80              }
81          }
82          else {
83              scanf("%d%d%d", &l[0], &l[1], &r[0], &r[1]);
84              l[0] ^= ans;
85              l[1] ^= ans;
86              r[0] ^= ans;
87              r[1] ^= ans;
88              ans = 0;
89
90              for (int i = 1; i <= tmp; i++)
91                  if (l[0] <= x[i][0] && l[1] <= x[i][1] &&
    ↳ x[i][0] <= r[0] && x[i][1] <= r[1])
92                      ans += w[i];
93
94              query(root);
95              printf("%d\n", ans);
96          }
97      }
98
99      return 0;
100 }

```

## 4.9 LCA最近公共祖先

### 4.9.1 Tarjan LCA $O(n + m)$

```

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

```

## 4.10 虚树

```

1  struct Tree {
2      vector<int> G[maxn], W[maxn];
3      int p[maxn], d[maxn], size[maxn], mn[maxn], mx[maxn];
4      bool col[maxn];
5      long long ans_sum;
6      int ans_min, ans_max;
7
8      void add(int x, int y, int z) {
9          G[x].push_back(y);
10         W[x].push_back(z);
11     }
12
13     void dfs(int x) {
14         size[x] = col[x];
15         mx[x] = (col[x] ? d[x] : -inf);
16         mn[x] = (col[x] ? d[x] : inf);
17
18         for (int i = 0; i < (int)G[x].size(); i++) {
19             d[G[x][i]] = d[x] + W[x][i];
20             dfs(G[x][i]);
21             ans_sum += (long long)size[x] * size[G[x][i]] *
    ↳ d[x];
22             ans_max = max(ans_max, mx[x] + mx[G[x][i]] -
    ↳ (d[x] << 1));
23             ans_min = min(ans_min, mn[x] + mn[G[x][i]] -
    ↳ (d[x] << 1));
24             size[x] += size[G[x][i]];
25             mx[x] = max(mx[x], mx[G[x][i]]);
26             mn[x] = min(mn[x], mn[G[x][i]]);
27         }
28     }
29
30     void clear(int x) {
31         G[x].clear();
32         W[x].clear();
33         col[x] = false;
34     }
35
36     void solve(int rt) {
37         ans_sum = 0;
38         ans_max = -inf;
39         ans_min = inf;

```

```

40     dfs(rt);
41     ans_sum <= 1;
42 }
43 } virtree;
44
45 void dfs(int);
46 int LCA(int, int);
47
48 vector<int> G[maxn];
49 int f[maxn][20], d[maxn], dfn[maxn], tim = 0;
50
51 bool cmp(int x, int y) {
52     return dfn[x] < dfn[y];
53 }
54
55 int n, m, lgn = 0, a[maxn], s[maxn], v[maxn];
56
57 int main() {
58     scanf("%d", &n);
59
60     for (int i = 1, x, y; i < n; i++) {
61         scanf("%d%d", &x, &y);
62         G[x].push_back(y);
63         G[y].push_back(x);
64     }
65
66     G[n + 1].push_back(1);
67     dfs(n + 1);
68
69     for (int i = 1; i <= n + 1; i++)
70         G[i].clear();
71
72     lgn--;
73
74     for (int j = 1; j <= lgn; j++)
75         for (int i = 1; i <= n; i++)
76             f[i][j] = f[f[i][j - 1]][j - 1];
77
78     scanf("%d", &m);
79
80     while (m--) {
81         int k;
82         scanf("%d", &k);
83
84         for (int i = 1; i <= k; i++)
85             scanf("%d", &a[i]);
86
87         sort(a + 1, a + k + 1, cmp);
88         int top = 0, cnt = 0;
89         s[++top] = v[++cnt] = n + 1;
90         long long ans = 0;
91
92         for (int i = 1; i <= k; i++) {
93             virtree.col[a[i]] = true;
94             ans += d[a[i]] - 1;
95             int u = LCA(a[i], s[top]);
96
97             if (s[top] != u) {
98                 while (top > 1 && d[s[top - 1]] >= d[u]) {
99                     virtree.add(s[top - 1], s[top],
100                                 d[s[top]] - d[s[top - 1]]);
101                     top--;
102                 }
103
104                 if (s[top] != u) {
105                     virtree.add(u, s[top], d[s[top]] -
106                                 d[u]);
107                     s[top] = v[++cnt] = u;
108                 }
109             }
110
111             s[++top] = a[i];

```

```

110     }
111
112     for (int i = top - 1; i; i--)
113         virtree.add(s[i], s[i + 1], d[s[i + 1]] -
114                     d[s[i]]);
115
116     virtree.solve(n + 1);
117     ans *= k - 1;
118     printf("%lld %d %d\n", ans - virtree.ans_sum,
119           virtree.ans_min, virtree.ans_max);
120
121     for (int i = 1; i <= k; i++)
122         virtree.clear(a[i]);
123     for (int i = 1; i <= cnt; i++)
124         virtree.clear(v[i]);
125
126     }
127
128     return 0;
129
130 void dfs(int x) {
131     dfn[x] = ++tim;
132     d[x] = d[f[x][0]] + 1;
133
134     while ((1 << lgn) < d[x])
135         lgn++;
136
137     for (int i = 0; i < (int)G[x].size(); i++)
138         if (G[x][i] != f[x][0]) {
139             f[G[x][i]][0] = x;
140             dfs(G[x][i]);
141         }
142 }
143
144 int LCA(int x, int y) {
145     if (d[x] != d[y]) {
146         if (d[x] < d[y])
147             swap(x, y);
148
149         for (int i = lgn; i >= 0; i--)
150             if (((d[x] - d[y]) >> i) & 1)
151                 x = f[x][i];
152     }
153
154     if (x == y)
155         return x;
156
157     for (int i = lgn; i >= 0; i--)
158         if (f[x][i] != f[y][i]) {
159             x = f[x][i];
160             y = f[y][i];
161         }
162
163     return f[x][0];
164 }

```

## 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
10 // 来存
11 // 或者开一个结构体维护倒过来的vector
12 void dfs(int x) {

```

```

11     h[x] = 1;
12
13     for (int y : G[x])
14         if (y != p[x]){
15             p[y] = x;
16             dfs(y);
17
18             if (h[y] > h[son[x]])
19                 son[x] = y;
20         }
21
22     if (!son[x]) {
23         v[x].push_back(1);
24         ans[x] = 0;
25         return;
26     }
27
28     h[x] = h[son[x]] + 1;
29     swap(v[x], v[son[x]]);
30
31     if (v[x][ans[son[x]]] == 1)
32         ans[x] = h[x] - 1;
33     else
34         ans[x] = ans[son[x]];
35
36     v[x].push_back(1);
37
38     int mx = v[x][ans[x]];
39     for (int y : G[x])
40         if (y != p[x] && y != son[x]) {
41             for (int j = 1; j <= h[y]; j++) {
42                 v[x][h[x] - j - 1] += v[y][h[y] - j];
43
44                 int t = v[x][h[x] - j - 1];
45                 if (t > mx || (t == mx && h[x] - j - 1 >
46                     ↪ ans[x])) {
47                     mx = t;
48                     ans[x] = h[x] - j - 1;
49                 }
50             }
51             v[y].clear();
52         }
53 }

```

```

23         f[0][y] = x;
24         d[y] = d[x] + 1;
25
26         dfs1(y);
27
28         mxd[x] = max(mxd[x], mxd[y]);
29         if (mxd[y] > mxd[son[x]])
30             son[x] = y;
31     }
32 }
33
34 // 第二遍dfs, 用于进行剖分和预处理梯子剖分(每条链向上延伸
35   ↪ 一倍)数组
36 void dfs2(int x) {
37     top[x] = (x == son[f[0][x]] ? top[f[0][x]] : x);
38
39     for (int y : G[x])
40         if (y != f[0][x])
41             dfs2(y);
42
43     if (top[x] == x) {
44         int u = x;
45         while (top[son[u]] == x)
46             u = son[u];
47
48         len[x] = d[u] - d[x];
49         for (int i = 0; i < len[x]; i++, u = f[0][u])
50             v[x].push_back(u);
51
52         u = x;
53         for (int i = 0; i < len[x] && u; i++, u = f[0][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] + k];
68 }

```

#### 4.11.1 梯子剖分

```

1 // 在线求一个点的第k祖先 O(n\log n)-O(1)
2 // 理论基础: 任意一个点x的k级祖先y所在长链长度一定>=k
3
4 // 全局数组定义
5 vector<int> G[maxn], v[maxn];
6 int d[maxn], mxd[maxn], son[maxn], top[maxn], len[maxn];
7 int f[19][maxn], log_tbl[maxn];
8
9 // 在主函数中两遍dfs之后加上如下预处理
10 log_tbl[0] = -1;
11 for (int i = 1; i <= n; i++)
12     log_tbl[i] = log_tbl[i / 2] + 1;
13 for (int j = 1; (1 << j) < n; j++)
14     for (int i = 1; i <= n; i++)
15         f[j][i] = f[j - 1][f[j - 1][i]];
16
17 // 第一遍dfs, 用于计算深度和找出重儿子
18 void dfs1(int x) {
19     mxd[x] = d[x];
20
21     for (int y : G[x])
22         if (y != f[0][x]){

```

## 4.12 左偏树

(参见k短路板子.)

## 4.13 莫队

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

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

### 4.13.1 回滚莫队(无删除莫队)(待完成)

### 4.13.2 莫队二次离线

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

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



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

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

剩余的三种情况(右端点左移或者移动左端点)都是类似的, 具体可以看代码.

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

```
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     return sa[x] + sb[bid[x] - 1];
41 }
42
43 void adds(int x) {
44     for (int k = 1; k <= bid[x]; k++)
45         sb[k]++;
46
47     for (int i = L[bid[x]]; i <= x; i++)
48         sa[i]++;
49 }
50
51 int querys(int x) {
52     if (x > global_n)
53         return 0; // 为了防止越界就判一下
54     return sa[x] + sb[bid[x] + 1];
55 }
56
57 vector<Q> vp[maxn], vs[maxn]; // prefix, suffix
58
59 long long fp[maxn], fs[maxn]; // prefix, suffix
```

```
58
59 int a[maxn], b[maxn];
60
61 long long ta[maxn], ans[maxn];
62
63 int main() {
64
65     int n, m;
66     scanf("%d%d", &n, &m);
67
68     global_n = n;
69
70     for (int i = 1; i <= n; i++)
71         scanf("%d", &a[i]);
72
73     memcpy(b, a, sizeof(int) * (n + 1));
74     sort(b + 1, b + n + 1);
75
76     for (int i = 1; i <= n; i++)
77         a[i] = lower_bound(b + 1, b + n + 1, a[i]) - b;
78
79     for (int i = 1; i <= n; i++) {
80         bid[i] = (i - 1) / B + 1;
81
82         if (!L[bid[i]])
83             L[bid[i]] = i;
84
85         R[bid[i]] = i;
86         cntb = bid[i];
87     }
88
89     for (int i = 1; i <= m; i++) {
90         scanf("%d%d", &q[i].l, &q[i].r);
91
92         q[i].d = bid[q[i].l];
93         q[i].id = i;
94     }
95
96     sort(q + 1, q + m + 1);
97
98     int l = 2, r = 1; // l, r是上一个询问的端点
99
100     for (int i = 1; i <= m; i++) {
101         int s = q[i].l, t = q[i].r; // s, t是当前要调整到的
102         ↪ 端点
103
104         if (s < l)
105             vs[r + 1].push_back(Q(s, l - 1, 1, i));
106         else if (s > l)
107             vs[r + 1].push_back(Q(l, s - 1, -1, i));
108
109         l = s;
110
111         if (t > r)
112             vp[l - 1].push_back(Q(r + 1, t, 1, i));
113         else if (t < r)
114             vp[l - 1].push_back(Q(t + 1, r, -1, i));
115
116         r = t;
117     }
118
119     for (int i = 1; i <= n; i++) { // 第一遍正着处理, 解决
120         ↪ 关于前缀的询问
121         fp[i] = fp[i - 1] + querys(a[i] + 1);
122
123         adds(a[i]);
124
125         for (auto q : vp[i]) {
126             long long tmp = 0;
127             for (int k = q.l; k <= q.r; k++)
```

```

126         tmp += querys(a[k] + 1);
127
128         ta[q.id] -= q.d * tmp;
129     }
130 }
131
132 memset(sa, 0, sizeof(sa));
133 memset(sb, 0, sizeof(sb));
134
135 for (int i = n; i; i--) { // 第二遍倒着处理, 解决关于后
    ↳ 缀的询问
136     fs[i] = fs[i + 1] + queryp(a[i] - 1);
137
138     addp(a[i]);
139
140     for (auto q : vs[i]) {
141         long long tmp = 0;
142         for (int k = q.l; k <= q.r; k++)
143             tmp += queryp(a[k] - 1);
144
145         ta[q.id] -= q.d * tmp;
146     }
147 }
148
149 l = 2;
150 r = 1;
151
152 for (int i = 1; i <= m; i++) { // 求出fs和fp之后再加上
    ↳ 这部分的贡献
153     int s = q[i].l, t = q[i].r;
154
155     ta[i] += fs[s] - fs[l];
156     ta[i] += fp[t] - fp[r];
157
158     l = s;
159     r = t;
160
161     ta[i] += ta[i - 1]; // 因为算出来的是相邻两个询问之
    ↳ 间的贡献, 所以要前缀和
162     ans[q[i].id] = ta[i];
163 }
164
165 for (int i = 1; i <= m; i++)
166     printf("%lld\n", ans[i]);
167
168 return 0;
169 }

```

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 + 5;
2
3 int n, bid[maxn], L[maxb], R[maxb], cntb;
4
5 struct DS { //  $O(\sqrt{n})$ 修改  $O(1)$ 查询
6     int total;
7     int sa[maxn], sb[maxb];
8
9     void init(const DS &o) {
10         total = o.total;
11         memcpy(sa, o.sa, sizeof(int) * (n + 1));
12         memcpy(sb, o.sb, sizeof(int) * (cntb + 1));
13     }
14
15     void add(int x) {
16         total++;
17         for (int k = 1; k <= bid[x]; k++)
18             sb[k]++;
19         for (int i = L[bid[x]]; i <= x; i++)
20             sa[i]++;
21     }
22
23     int querys(int x) {
24         if (x > n)
25             return 0;
26
27         return sb[bid[x] + 1] + sa[x];
28     }
29
30     int queryp(int x) {
31         return total - querys(x + 1);
32     }
33 } pr[maxb];
34
35 int c[maxn]; // 树状数组
36
37 void addc(int x, int d) {
38     while (x) {
39         c[x] += d;
40         x -= x & -x;
41     }
42 }
43
44 int queryc(int x) {
45     int ans = 0;
46     while (x <= n) {
47         ans += c[x];
48         x += x & -x;
49     }
50     return ans;
51 }
52
53 long long fp[maxn], fs[maxn];
54
55 int rnk[maxn], val[maxn][B + 5];
56
57 long long dat[maxb][maxb];
58
59 int a[maxn];
60
61 int main() {
62
63     int m;
64     cin >> n >> m;

```

#### 4.13.3 带修莫队在线化 $O(n^{\frac{5}{3}})$

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

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

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

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

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

#### 4.13.4 莫队二次离线 在线化 $O((n + m)\sqrt{n})$

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

按照莫队二次离线的转移方法操作之后发现只剩两边散块的贡献没有解决. 这时可以具体问题具体解决, 例如求逆序对的话直接预处理

```

65
66 for (int i = 1; i <= n; i++) {
67     cin >> a[i];
68
69     bid[i] = (i - 1) / B + 1;
70     if (!L[bid[i]])
71         L[bid[i]] = i;
72     R[bid[i]] = i;
73     cntb = bid[i];
74
75     rnk[i] = i;
76 }
77
78 for (int k = 1; k <= cntb; k++)
79     sort(rnk + L[k], rnk + R[k] + 1, [](int x, int y)
80         ↪ {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 - 1] -
85             ↪ 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); // 单块内的逆序
93         ↪ 对直接用树状数组预处理
94         addc(a[i], 1);
95     }
96
97     for (int i = L[k]; i <= R[k]; i++)
98         addc(a[i], -1);
99 }
100
101 for (int i = 1; i <= n; i++) {
102     if (i > 1 && i == L[bid[i]])
103         pr[bid[i]].init(pr[bid[i] - 1]);
104
105     fp[i] = fp[i - 1] + pr[bid[i]].querys(a[i] + 1);
106
107     pr[bid[i]].add(a[i]);
108 }
109
110 for (int i = n; i; i--) {
111     fs[i] = fs[i + 1] + (n - i - queryc(a[i] + 1));
112     addc(a[i], 1);
113 }
114
115 for (int s = 1; s <= cntb; s++)
116     for (int t = s + 1; t <= cntb; t++) {
117         dat[s][t] = dat[s][t - 1] + dat[t][t];
118
119         for (int i = L[t]; i <= R[t]; i++) // 块间的逆
120             ↪ 序对用刚才处理的分块求出
121             dat[s][t] += pr[t - 1].querys(a[i] + 1) -
122                 ↪ pr[s - 1].querys(a[i] + 1);
123     }
124
125 long long ans = 0;
126
127 while (m--) {
128     long long s, t;
129     cin >> s >> t;
130
131     int l = s ^ ans, r = t ^ ans;
132
133     if (bid[l] == bid[r])

```

```

129     ans = val[l][r - 1 + 1];
130     else {
131         ans = dat[bid[l] + 1][bid[r] - 1];
132
133         ans += fp[r] - fp[L[bid[r]] - 1];
134         for (int i = L[bid[r]]; i <= r; i++)
135             ans -= pr[bid[l]].querys(a[i] + 1);
136
137         ans += fs[l] - fs[R[bid[l]] + 1];
138         for (int i = l; i <= R[bid[l]]; i++)
139             ans -= (a[i] - 1) - pr[bid[r] -
140                 ↪ 1].queryp(a[i] - 1);
141
142         int i = L[bid[l]], j = L[bid[r]], w = 0; // 手
143             ↪ 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]] :
155                 ↪ (int)1e9), y = (j <= R[bid[r]] ?
156                 ↪ a[rnk[j]] : (int)1e9);
157
158             if (x < y) {
159                 ans += w;
160                 i++;
161             }
162             else {
163                 j++;
164                 w++;
165             }
166         }
167
168         cout << ans << '\n';
169     }
170 }
171
172 return 0;
173 }

```

## 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         }
29     }
30 }

```

```

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
47         a[i] = j;
48         k = i;
49     }
50 }
51 }

```

### 5.2 AC自动机

```

1 int ch[maxn][26], f[maxn][26], q[maxn], sum[maxn], cnt = 0;
2
3 // 在字典树中插入一个字符串  $O(n)$ 
4 int insert(const char *c) {
5     int x = 0;
6     while (*c) {
7         if (!ch[x][*c - 'a'])
8             ch[x][*c - 'a'] = ++cnt;
9         x = ch[x][*c++ - 'a'];
10    }
11    return x;
12 }
13
14 // 建AC自动机  $O(n * \sigma)$ 
15 void getfail() {
16     int x, head = 0, tail = 0;
17
18     for (int c = 0; c < 26; c++)
19         if (ch[0][c])
20             q[tail++] = ch[0][c]; // 把根节点的儿子加入队列
21
22     while (head != tail) {
23         x = q[head++];
24
25         G[f[x][0]].push_back(x);
26         fill(f[x] + 1, f[x] + 26, cnt + 1);
27
28         for (int c = 0; c < 26; c++) {
29             if (ch[x][c]) {
30                 int y = f[x][0];
31
32                 f[ch[x][c]][0] = ch[y][c];
33                 q[tail++] = ch[x][c];
34             }
35             else
36                 ch[x][c] = ch[f[x][0]][c];
37         }
38     }
39     fill(f[0], f[0] + 26, cnt + 1);
40 }

```

## 5.3 后缀数组

### 5.3.1 倍增

```

1 constexpr int maxn = 100005;
2
3 void get_sa(char *s, int n, int *sa, int *rnk, int *height)
4     ↪ { // 1-base
5     static int buc[maxn], id[maxn], p[maxn], t[maxn * 2];
6
7     int m = 300;
8
9     for (int i = 1; i <= n; i++)
10         buc[rnk[i] = s[i]]++;
11     for (int i = 1; i <= m; i++)
12         buc[i] += buc[i - 1];
13     for (int i = n; i; i--)
14         sa[buc[rnk[i]]--] = i;
15
16     memset(buc, 0, sizeof(int) * (m + 1));
17
18     for (int k = 1, cnt = 0; cnt != n; k *= 2, m = cnt) {
19         cnt = 0;
20         for (int i = n; i > n - k; i--)
21             id[++cnt] = i;
22
23         for (int i = 1; i <= n; i++)
24             if (sa[i] > k)
25                 id[++cnt] = sa[i] - k;
26
27         for (int i = 1; i <= n; i++)
28             buc[p[i] = rnk[id[i]]]++;
29         for (int i = 1; i <= m; i++)
30             buc[i] += buc[i - 1];
31         for (int i = n; i; i--)
32             sa[buc[p[i]]--] = id[i];
33
34         memset(buc, 0, sizeof(int) * (m + 1));
35
36         memcpy(t, rnk, sizeof(int) * (max(n, m) + 1));
37
38         cnt = 0;
39         for (int i = 1; i <= n; i++) {
40             if (t[sa[i]] != t[sa[i - 1]] || t[sa[i] + k] !=
41                 ↪ t[sa[i - 1] + k])
42                 cnt++;
43
44             rnk[sa[i]] = cnt;
45         }
46
47         for (int i = 1; i <= n; i++)
48             sa[rnk[i]] = i;
49
50         for (int i = 1, k = 0; i <= n; i++) { // 顺便求height
51             if (k)
52                 k--;
53
54             while (s[i + k] == s[sa[rnk[i] - 1] + k])
55                 k++;
56
57             height[rnk[i]] = k; // height[i] = lcp(sa[i], sa[i
58                 ↪ - 1])
59         }
60
61         char s[maxn];
62         int sa[maxn], rnk[maxn], height[maxn];
63     }
64
65 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, 如果其他部分
2     ↪ 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] == l_type;
9 }
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
23 // 诱导排序(从*型诱导到L型,从L型诱导到S型)
24 // 调用之前应将*型按要求放入SA中
25 void induced_sort(int *s, int *sa, int *tp, int *buc, int
26     ↪ *lbuc, int *sbuc, int n, int m) {
27     for (int i = 0; i <= n; i++)
28         if (sa[i] > 0 && tp[sa[i] - 1] == l_type)
29             sa[lbuc[sa[i] - 1]++] = sa[i] - 1;
30
31     for (int i = 1; i <= m; i++)
32         sbuc[i] = buc[i] - 1;
33
34     for (int i = n; ~i; i--)
35         if (sa[i] > 0 && tp[sa[i] - 1] == s_type)
36             sa[sbuc[sa[i] - 1]--] = sa[i] - 1;
37 }
38
39 // s是输入字符串, n是字符串的长度, m是字符集的大小
40 int *sais(int *s, int len, int m) {
41     int n = len - 1;
42
43     int *tp = new int[n + 1];
44     int *pos = new int[n + 1];
45     int *name = new int[n + 1];
46     int *sa = new int[n + 1];
47     int *buc = new int[m + 1];
48     int *lbuc = new int[m + 1];
49     int *sbuc = new int[m + 1];
50
51     memset(buc, 0, sizeof(int) * (m + 1));
52     memset(lbuc, 0, sizeof(int) * (m + 1));

```



```

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

```

```

119  lbuc[0] = sbuc[0] = 0;
120  for (int i = 1; i <= m; i++) {
121      lbuc[i] = buc[i - 1];
122      sbuc[i] = buc[i] - 1;
123  }
124
125  memset(sa, -1, sizeof(int) * (n + 1));
126  for (int i = cnt - 1; ~i; i--)
127      sa[sbuc[s[pos[tsa[i]]]--]] = pos[tsa[i]];
128  induced_sort(s, sa, tp, buc, lbuc, sbuc, n, m);
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     ↪ {
146     static int a[maxn];
147
148     for (int i = 1; i <= n; i++)
149         a[i - 1] = s[i];
150
151     a[n] = '$';
152
153     int *t = sais(a, n + 1, 256);
154     memcpy(sa, t, sizeof(int) * (n + 1));
155     delete[] t;
156
157     sa[0] = 0;
158     for (int i = 1; i <= n; i++)
159         rnk[++sa[i]] = i;
160
161     for (int i = 1, k = 0; i <= n; i++) { // 求height
162         if (k)
163             k--;
164
165         while (s[i + k] == s[sa[rnk[i] - 1] + k])
166             k++;
167
168         height[rnk[i]] = k; // height[i] = Lcp(sa[i], sa[i
169             ↪ - 1])
170     }
171 }

```

### 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 =
4      ↪ 0;
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;
58 }

```

```

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 }

```

## 5.4 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或者
2   ↪ 哈希表替换
3 // 注意!!!结点数要开成串长的两倍
4
5 // 全局变量与数组定义
6 int last, val[maxn], par[maxn], go[maxn][26], sam_cnt;
7 int c[maxn], q[maxn]; // 用来桶排序

```

### 5.5.1 广义后缀自动机

下面的写法复杂度是 $\Sigma$ 串长的，但是胜在简单。

如果建字典树然后BFS建自动机可以做到 $O(n|\Sigma|)$ ( $n$ 是字典树结点数)，但是后者写起来比较麻烦。

```

1 int extend(int p, int c) {
2     int np = 0;
3
4     if (!go[p][c]) {
5         np = ++sam_cnt;
6         val[np] = val[p] + 1;
7         while (p && !go[p][c]) {
8             go[p][c] = np;
9             p = par[p];
10        }
11    }
12
13    if (!p)
14        par[np] = 1;
15    else {
16        int q = go[p][c];
17
18        if (val[q] == val[p] + 1)
19            par[np] = q;
20        else {
21            int nq = ++sam_cnt;
22            val[nq] = val[p] + 1;
23            memcpy(go[nq], go[q], sizeof(go[q]));
24
25            par[nq] = par[q];
26            par[np] = par[q] = nq;
27
28            while (p && go[p][c] == q) {
29                go[p][c] = nq;
30                p = par[p];
31            }
32        }
33    }
34
35    last = np;
36 }

```

```

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 // 调用时直接last = 1然后一路调用last = extend(last,
    ↪ c)就行了

```

```

16     x -> lazy = true;
17
18     update(x -> val - val[x -> r] + 1, x -> val -
    ↪ val[par[x -> l]], 1);
19
20     x -> ch[1] = y;
21
22     (y = x) -> refresh();
23
24     x = x -> p;
25 }
26
27 return y;
28 }
29
30 // 以下是main函数中的用法
31 for (int i = 1; i <= n; i++) {
32     tim++;
33     access(null + id[i]);
34
35     if (i >= m) // 例题询问长度是固定的，如果不固定的话就
    ↪ 按照右端点离线即可
36         ans[i - m + 1] = query(i - m + 1, i);
37 }

```

## 5.6 回文树

### 5.5.2 区间本质不同子串计数(后缀自动机+LCT+线段树)

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

做法: 考虑建出后缀自动机, 然后枚举右端点, 用线段树维护每个左端点的答案.

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

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

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

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

时间复杂度 $O(n \log^2 n)$ , 空间 $O(n)$ , 当然如果强制在线的话, 就把线段树改成主席树, 空间复杂度就和时间复杂度同阶了.

```

1 int tim; // tim实际上就是当前的右端点
2
3 node *access(node *x) {
4     node *y = null;
5
6     while (x != null) {
7         splay(x);
8
9         x -> ch[1] = null;
10        x -> refresh();
11
12        if (x -> val) // val记录的是上次访问时间, 也就
    ↪ 是right集合最大值
13            update(x -> val - val[x -> r] + 1, x -> val -
    ↪ val[par[x -> l]], -1);
14
15        x -> val = tim;

```

```

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     u = x;
74     for (int i = 0; i < len[x]; i++) { // 梯子剖分, 要延
75         ↳ 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] + k];
91 }
92
93 char get_char(int x, int k) { // 查询x前面k个的字符是哪个
94     return chr[get_anc(x, k)];
95 }
96
97 int getfail(int x, int p) {
98     if (get_char(x, val[p] + 1) == chr[x])
99         return p;
100     return fail[p][chr[x] - 'a'];
101 }
102
103 int extend(int x) {
104
105     int p = pam_last[f[0][x]], c = chr[x] - 'a';
106
107     p = getfail(x, p);
108
109     int new_last;
110
111     if (!go[p][c]) {
112         int q = ++pam_cnt, now = p;
113         val[q] = val[p] + 2;
114
115         p = getfail(x, par[p]);
116
117         par[q] = go[p][c];
118         new_last = go[now][c] = q;
119
120         for (int i = 0; i < 26; i++)
121             fail[q][i] = fail[par[q]][i];
122
123         if (get_char(x, val[par[q]]) >= 'a')
124             fail[q][get_char(x, val[par[q]]) - 'a'] =
125                 ↳ par[q];
126
127         if (val[q] <= n)
128             weight[q] = (weight[par[q]] + (long long)(n -
129                 ↳ val[q] + 1) * pow_26[n - val[q]]) % mod;
130         else
131             weight[q] = weight[par[q]];
132     }
133     else
134         new_last = go[p][c];
135
136     pam_last[x] = new_last;

```

```

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

```

```

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

```

## 5.7 Manacher马拉车

```

1 //n为串长,回文半径输出到p数组中
2 //数组要开串长的两倍
3 void manacher(const char *t, int n) {
4     static char s[maxn * 2];
5
6     for (int i = n; i; 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) : 1);
19         while (s[i - p[i]] == s[i + p[i]])
20             p[i]++;
21
22         if (i + p[i] > mx) {
23             mx = i + p[i];
24             j = i;

```

```
25 |   |   }
26 |   |   }
27 |   |   }
```

### 5.8 字符串原理

KMP和AC自动机的fail指针存储的都是它在串或者字典树上的最长后缀，因此要判断两个前缀是否互为后缀时可以直接用fail指针判断. 当然它不能做子串问题，也不能做最长公共后缀.

后缀数组利用的主要是LCP长度可以按照字典序做RMQ的性质，与某个串的LCP长度 $\geq$ 某个值的后缀形成一个区间. 另外一个比较好的性质是本质不同的子串个数 = 所有子串数 - 字典序相邻的串的height.

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

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



## 6 动态规划

### 6.1 决策单调性 $O(n \log n)$

```

1 int a[maxn], q[maxn], p[maxn], g[maxn]; // 存左端点, 右端点
   ↳ 就是下一个左端点 - 1
2
3 long long f[maxn], s[maxn];
4
5 int n, m;
6
7 long long calc(int l, int r) {
8     if (r < l)
9         return 0;
10
11     int mid = (l + r) / 2;
12     if ((r - l + 1) % 2 == 0)
13         return (s[r] - s[mid]) - (s[mid] - s[l - 1]);
14     else
15         return (s[r] - s[mid]) - (s[mid - 1] - s[l - 1]);
16 }
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) < f[i]) {
32                 f[i] = f[q[head]] + calc(q[head] + 1, i);
33                 g[i] = g[q[head]] + 1;
34             }
35             while (head < tail && p[head + 1] <= i + 1)
36                 head++;
37             if (head <= tail)
38                 p[head] = i + 1;
39         }
40         f[i] += tmp;
41
42         int r = n;
43
44         while(head <= tail) {
45             if (f[q[tail]] + calc(q[tail] + 1, p[tail]) >
46                 ↳ f[i] + calc(i + 1, p[tail])) {
47                 r = p[tail] - 1;
48                 tail--;
49             }
50             else if (f[q[tail]] + calc(q[tail] + 1, r) <=
51                 ↳ f[i] + calc(i + 1, r)) {
52                 if (r < n) {
53                     q[++tail] = i;
54                     p[tail] = r + 1;
55                 }
56                 break;
57             }
58             else {
59                 int L = p[tail], R = r;
60                 while (L < R) {
61                     int M = (L + R) / 2;
62
63                     if (f[q[tail]] + calc(q[tail] + 1, M)
64                         ↳ <= f[i] + calc(i + 1, M))
65                         L = M + 1;
66                     else

```

```

64         R = M;
65     }
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
79 return g[n];
80 }

```

## 7 杂项

### 7.1 $O(1)$ 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，非必要不建议使
  ↳ 用
4 long long mul(long long a, long long b, long long p) {
5     a %= p;
6     b %= p;
7     return ((a * b - p * (long long)((long double)a / p * b
  ↳ + 0.5)) % p + p) % p;
8 }
9
10 // 指令集快速乘
11 // 试机记得测试能不能过编译
12 inline long long mul(const long long a, const long long b,
  ↳ const long long p) {
13     long long ans;
14     __asm__ __volatile__ ("tmulq %%rbx\n\tdivq %%rcx\n" :
  ↳ "=d"(ans) : "a"(a), "b"(b), "c"(p));
15     return ans;
16 }
17
18 // int乘法取模，大概比直接做快一倍
19 inline int mul_mod(int a, int b, int p) {
20     int ans;
21     __asm__ __volatile__ ("tmull %%ebx\n\tdivl %%ecx\n" :
  ↳ "=d"(ans) : "a"(a), "b"(b), "c"(p));
22     return ans;
23 }
```

### 7.2 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 }
```

### 7.3 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，50.57
10 x = decimal.Decimal('50.5679'). \
11     quantize(decimal.Decimal('0.00'), decimal.ROUND_HALF_UP)
  ↳ # 四舍五入
12 # 第二个参数可选如下：
13 # 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 rounding
  ↳ towards zero would have been 0 or 5; otherwise towards
  ↳ zero)
21
22 print('%f', x) # 这样做只有float的精度
23 s = str(x)
24
25 decimal.is_finite(x) # x是否有穷(NaN也算)
26 decimal.is_infinite(x)
27 decimal.is_nan(x)
28 decimal.is_normal(x) # x是否正常
29 decimal.is_signed(x) # 是否为负数
30
31 decimal.fma(a, b, c) # a * b + c, 精度更高
32
33 x.exp(), x.ln(), x.sqrt(), x.log10()
34
35 # 可以转复数，前提是要import complex
```

### 7.4 $O(n^2)$ 高精度

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

```

45     a[i] = s[a[0] - i] - '0';
46
47     while (a[0] && !a[a[0]])
48         a[0]--;
49 }
50
51 void input() {
52     string s;
53     cin >> s;
54     *this = s;
55 }
56
57 string str() const {
58     if (!a[0])
59         return "0";
60
61     string s;
62     if (negative)
63         s = "-";
64
65     for (int i = a[0]; i; i--)
66         s.push_back('0' + a[i]);
67
68     return s;
69 }
70
71 operator string () const {
72     return str();
73 }
74
75 big_decimal operator - () const {
76     big_decimal o = *this;
77     if (a[0])
78         o.negative ^= true;
79
80     return o;
81 }
82
83 friend big_decimal abs(const big_decimal &u) {
84     big_decimal o = u;
85     o.negative = false;
86     return o;
87 }
88
89 big_decimal &operator <= (int k) {
90     a[0] += k;
91
92     for (int i = a[0]; i > k; i--)
93         a[i] = a[i - k];
94
95     for(int i = k; i; i--)
96         a[i] = 0;
97
98     return *this;
99 }
100
101 friend big_decimal operator << (const big_decimal &u,
102     ↪ 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 &u,
122     ↪ int k) {
123     big_decimal o = u;
124     return o >= k;
125 }
126
127 friend int cmp(const big_decimal &u, const big_decimal
128     ↪ &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
140     if (u.a[0] != v.a[0])
141         return u.a[0] < v.a[0] ? -1 : 1;
142
143     for (int i = u.a[0]; i; i--)
144         if (u.a[i] != v.a[i])
145             return u.a[i] < v.a[i] ? -1 : 1;
146
147     return 0;
148 }
149
150 friend bool operator < (const big_decimal &u, const
151     ↪ big_decimal &v) {
152     return cmp(u, v) == -1;
153 }
154
155 friend bool operator > (const big_decimal &u, const
156     ↪ big_decimal &v) {
157     return cmp(u, v) == 1;
158 }
159
160 friend bool operator <= (const big_decimal &u, const
161     ↪ big_decimal &v) {
162     return cmp(u, v) <= 0;
163 }
164
165 friend bool operator >= (const big_decimal &u, const
166     ↪ big_decimal &v) {
167     return cmp(u, v) >= 0;
168 }
169
170 friend big_decimal decimal_plus(const big_decimal &u,
171     ↪ const big_decimal &v) { // 保证u, v均为正数的话可以
172     ↪ 直接调用
173     big_decimal o;
174
175     o.a[0] = max(u.a[0], v.a[0]);
176
177     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 &u,
    ↪ const big_decimal &v) { // 保证u, v均为正数的话可以
    ↪ 直接调用
188     int k = cmp(u, v);
189
190     if (k == -1)
191         return -decimal_minus(v, u);
192     else if (k == 0)
193         return big_decimal(0);
194
195     big_decimal o;
196
197     o.a[0] = u.a[0];
198
199     for (int i = 1; i <= u.a[0]; i++) {
200         o.a[i] += u.a[i] - v.a[i];
201
202         if (o.a[i] < 0) {
203             o.a[i] += 10;
204             o.a[i + 1]--;
205         }
206     }
207
208     while (o.a[0] && !o.a[o.a[0]])
209         o.a[0]--;
210
211     return o;
212 }
213
214 friend big_decimal decimal_multi(const big_decimal &u,
    ↪ const big_decimal &v) {
215     big_decimal o;
216
217     o.a[0] = u.a[0] + v.a[0] - 1;
218
219     for (int i = 1; i <= u.a[0]; i++)
220         for (int j = 1; j <= v.a[0]; j++)
221             o.a[i + j - 1] += u.a[i] * v.a[j];
222
223     for (int i = 1; i <= o.a[0]; i++)
224         if (o.a[i] >= 10) {
225             o.a[i + 1] += o.a[i] / 10;
226             o.a[i] %= 10;
227         }
228
229     if (o.a[o.a[0] + 1])
230         o.a[0]++;
231
232     return o;
233 }
234
235 friend pair<big_decimal, big_decimal>
    ↪ decimal_divide(big_decimal u, big_decimal v) { //
    ↪ 整除
236     if (v > u)
237         return make_pair(big_decimal(0), u);

```

```

238
239     big_decimal o;
240     o.a[0] = u.a[0] - v.a[0] + 1;
241
242     int m = v.a[0];
243     v <= u.a[0] - m;
244
245     for (int i = u.a[0]; i >= m; i--) {
246         while (u >= v) {
247             u = u - v;
248             o.a[i - m + 1]++;
249         }
250
251         v >= 1;
252     }
253
254     while (o.a[0] && !o.a[o.a[0]])
255         o.a[0]--;
256
257     return make_pair(o, u);
258 }
259
260 friend big_decimal operator + (const big_decimal &u,
    ↪ const big_decimal &v) {
261     if (u.negative || v.negative) {
262         if (u.negative && v.negative)
263             return -decimal_plus(-u, -v);
264
265         if (u.negative)
266             return v - (-u);
267
268         if (v.negative)
269             return u - (-v);
270     }
271
272     return decimal_plus(u, v);
273 }
274
275 friend big_decimal operator - (const big_decimal &u,
    ↪ const big_decimal &v) {
276     if (u.negative || v.negative) {
277         if (u.negative && v.negative)
278             return -decimal_minus(-u, -v);
279
280         if (u.negative)
281             return -decimal_plus(-u, v);
282
283         if (v.negative)
284             return decimal_plus(u, -v);
285     }
286
287     return decimal_minus(u, v);
288 }
289
290 friend big_decimal operator * (const big_decimal &u,
    ↪ const big_decimal &v) {
291     if (u.negative || v.negative) {
292         big_decimal o = decimal_multi(abs(u), abs(v));
293
294         if (u.negative ^ v.negative)
295             return -o;
296         return o;
297     }
298
299     return decimal_multi(u, v);
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> decimal_div(const
329     ↪ big_decimal &u, const big_decimal &v) {
330     if (u.negative || v.negative) {
331         pair<big_decimal, big_decimal> o =
332             ↪ decimal_div(abs(u), abs(v));
333
334         if (u.negative ^ v.negative)
335             return make_pair(-o.first, -o.second);
336         return o;
337     }
338
339     return decimal_divide(u, v);
340 }
341
342 friend big_decimal operator / (const big_decimal &u,
343     ↪ const big_decimal &v) { // v不能是0
344     if (u.negative || v.negative) {
345         big_decimal o = abs(u) / abs(v);
346
347         if (u.negative ^ v.negative)
348             return -o;
349         return o;
350     }
351
352     return decimal_divide(u, v).first;
353 }
354
355 friend big_decimal operator % (const big_decimal &u,
356     ↪ const big_decimal &v) {
357     if (u.negative || v.negative) {
358         big_decimal o = abs(u) % abs(v);
359
360         if (u.negative ^ v.negative)
361             return -o;
362         return o;
363     }
364
365     return decimal_divide(u, v).second;
366 }
367 };

```

## 7.5 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 (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 }

```

## 7.6 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.

## 7.7 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
<b>998244353</b>	119	23	3
<b>1004535809</b>	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.

## 7.8 xorshift

```

1  ull k1, k2;
2  const int mod = 10000000;
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 }

```

## 7.9 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 }

```

## 7.10 STL

### 7.10.1 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)`

### 7.10.2 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

### 7.10.3 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.

## 7.11 Public Based DataStructure(PB\_DS)

### 7.11.1 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)

```

### 7.11.2 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](http://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

### 7.11.3 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.)

## 7.12 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

```

## 7.13 其他C++相关

### 7.13.1 <cmath>

- `std::log1p(x)`: (注意是数字1)返回 $\ln(1+x)$ 的值,  $x$ 非常接近0时比直接`exp`精确得多.
- `std::hypot(x, y[, z])`: 返回平方和的平方根, 或者说到原点的欧几里德距离.

### 7.13.2 <algorithm>

- `std::all_of(begin, end, f)`: 检查范围内元素调用函数f后是否全返回真. 类似地还有`std::any_of`和`std::none_of`.
- `std::for_each(begin, end, f)`: 对范围内所有元素调用一次f. 如果传入的是引用, 也可以用f修改. (例如`for_each(a, a + n, [](int &x){cout << ++x << "\n";})`)
- `std::for_each_n(begin, n, f)`: 同上, 只不过范围改成了从begin开始的n个元素.
- `std::copy()`, `std::copy_n()`: 用法谁都会, 但标准里说如果元素是可平凡复制的(比如int), 那么它会避免批量赋值, 并且调用`std::memmove()`之类的快速复制函数. (一句话总结: 它跑得快)
- `std::rotate(begin, mid, end)`: 循环移动, 移动后mid位置的元素会跑到first位置. C++11起会返回begin位置的元素移动后的位置.
- `std::unique(begin, end)`: 去重, 返回去重后的end.
- `std::partition(begin, end, f)`: 把f为true的放在前面, false的放在后面, 返回值是第二部分的开头, **不保持相对顺序**. 如果要保留相对顺序可以用`std::stable_partition()`, 比如写整体二分.
- `std::partition_copy(begin, end, begin_t, begin_f, f)`: 不修改原数组, 把true的扔到begin\_t, false的扔到begin\_f. 返回值是两部分结尾的迭代器的pair.
- `std::equal_range(begin, end, x)`: 在已经排好序的数组里找到等于x的范围.

- `std::minmax(a, b)`: 返回`pair(min(a, b), max(a, b))`. 比如`tie(l, r) = minmax(l, r)`.

### 7.13.3 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)`.

### 7.13.4 <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.

## 7.14 一些游戏

### 7.14.1 炉石传说

两个随从 $(a_i, h_i)$ 和 $(a_j, h_j)$ 皇城PK, 最后只有 $a_i \times h_i$ 较大的一方才有可能活下来, 当然也有可能一起死.

## 7.15 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.

### 7.15.1 计数相关

1. **卡特兰数(A000108)**  
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, ...  
性质见“数学”部分.
2. **(大)施罗德数(A006318)**  
1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, 5293446, 27297738, 142078746, 745387038, ... (0-based)  
性质同样见“数学”部分, 和卡特兰数放在一起.
3. **小施罗德数(A001003)**  
1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859, 2646723, 13648869, 71039373, 372693519, ... (0-based)  
性质位置同上.  
小施罗德数除了第0项以外都是施罗德数的一半.
4. **默慈金数(Motzkin numbers, A001006)**  
1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634, 310572, 853467, 2356779, ... (0-based)  
性质位置同上.
5. **将点按顺序排成一圈后不自交的树的个数(A001764)**  
1, 1, 3, 12, 55, 273, 1428, 7752, 43263, 246675, 1430715, 8414640, 50067108, 300830572, 1822766520, ... (0-based)  
$$a_n = \frac{\binom{3n}{n}}{2n+1}$$
也就是说, 在圆上按顺序排列的 $n$ 个点之间连 $n-1$ 条不相交(除端点外)的弦, 组成一棵树的方案数.  
也等于每次只能向右或向上, 并且不能高于 $y = 2x$ 这条直线, 从 $(0, 0)$ 走到 $(n, 2n)$ 的方案数.  
扩展: 如果改成不能高于 $y = kx$ 这条直线, 走到 $(n, kn)$ 的方案数, 那么答案就是 $\frac{\binom{(k+1)n}{n}}{kn+1}$ .

6.  **$n$ 个点的圆上画不相交的弦的方案数(A054726)**  
1, 1, 2, 8, 48, 352, 2880, 25216, 231168, 2190848, 21292032, 211044352, 2125246464, 21681954816, ... (0-based)  
 $a_n = 2^n s_{n-2}$  ( $n > 2$ ),  $s_n$ 是上面的小施罗德数.  
和上面的区别在于, 这里可以不连满 $n - 1$ 条边. 另外默慈金数画的弦不能共享端点, 但是这里可以.
7. **Wedderburn-Etherington numbers(A001190)**  
0, 1, 1, 1, 2, 3, 6, 11, 23, 46, 98, 207, 451, 983, 2179, 4850, 10905, 24631, 56011, 127912, 293547, ... (0-based)  
每个结点都有0或者2个儿子, 且总共有 $n$ 个叶子结点的二叉树方案数. (无标号)  
同时也是 $n - 1$ 个结点的无标号二叉树个数.  
$$A(x) = x + \frac{A(x)^2 + A(x^2)}{2} = 1 - \sqrt{1 - 2x - A(x^2)}$$
8. **划分数(A000041)**  
1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, 135, 176, 231, 297, 385, 490, 627, 792, 1002, ... (0-based)
9. **贝尔数(A000110)**  
1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, 1382958545, ... (0-based)
10. **错位排列数(A0000166)**  
1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, 2290792932, 32071101049, ... (0-based)
11. **交替阶乘(A005165)**  
0, 1, 1, 5, 19, 101, 619, 4421, 35899, 326981, 3301819, 36614981, 442386619, 5784634181, 81393657019, ...  
$$n! - (n - 1)! + (n - 2)! - \dots 1! = \sum_{i=0}^{n-1} (-1)^i (n - i)!.$$
  
 $a_0 = 0, a_n = n! - a_{n-1}.$

7.15.2 线性递推数列

1. **Lucas数(A000032)**  
2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349, 15127, ...
2. **斐波那契数(A000045)**  
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ...
3. **泰波那契数(Tribonacci, A000071)**  
0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705, 3136, 5768, 10609, 19513, 35890, ...  
 $a_0 = a_1 = 0, a_2 = 1, a_n = a_{n-1} + a_{n-2} + a_{n-3}.$
4. **Pell数(A0000129)**  
0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461, 80782, 195025, 470832, 1136689, ...  
 $a_0 = 0, a_1 = 1, a_n = 2a_{n-1} + a_{n-2}.$
5. **帕多万(Padovan)数(A0000931)**  
1, 0, 0, 1, 0, 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49, 65, 86, 114, 151, 200, 265, 351, 465, 616, 816, 1081, 1432, 1897, 2513, 3329, 4410, 5842, 7739, 10252, 13581, 17991, 23833, 31572, ...  
 $a_0 = 1, a_1 = a_2 = 0, a_n = a_{n-2} + a_{n-3}.$
6. **Jacobsthal numbers(A001045)**  
0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, 683, 1365, 2731, 5461, 10923, 21845, 43691, 87381, 174763, ...  
 $a_0 = 0, a_1 = 1. a_n = a_{n-1} + 2a_{n-2}$   
同时也是最接近 $\frac{2^n}{3}$ 的整数.
7. **佩林数(A001608)**  
3, 0, 2, 3, 2, 5, 5, 7, 10, 12, 17, 22, 29, 39, 51, 68, 90, 119, 158, 209, 277, 367, 486, 644, 853, ...  
 $a_0 = 3, a_1 = 0, a_2 = 2, a_n = a_{n-2} + a_{n-3}$

7.15.3 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.

7.15.4 其他

1. **伯努利数(A027641)**  
见“数学/常见数列”部分.
2. **四个柱子的汉诺塔(A007664)**  
0, 1, 3, 5, 9, 13, 17, 25, 33, 41, 49, 65, 81, 97, 113, 129, 161, 193, 225, 257, 289, 321, 385, 449, ...  
差分之后可以发现其实就是1次+1, 2次+2, 3次+4, 4次+8...的规律.
3. **乌拉姆数(Ulam numbers, A002858)**  
1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, 62, 69, 72, 77, 82, 87, 97, 99, 102, 106, 114, 126, 131, 138, 145, 148, 155, 175, 177, 180, 182, 189, 197, 206, 209, 219, 221, 236, 238, 241, 243, 253, 258, 260, 273, 282, 309, 316, 319, 324, 339 ...  
 $a_1 = 1, a_2 = 2, a_n$ 表示在所有 $> a_{n-1}$ 的数中, 最小的, 能被表示成(前面的两个不同的元素的和)的数.

7.16 编译选项

- -O2 -g -std=c++14: 狗都知道
- -Wall -Wextra -Wshadow -Wconversion: 更多警告
  - Werror: 强制将所有Warning变成Error
- -fsanitize=(address/undefined): 检查有符号整数溢出(算ub)/数组越界
  - 注意无符号类型溢出不算ub.
- -fno-ms-extensions: 关闭一些和msvc保持一致的特性, 例如, 不标返回值类型的函数会报CE而不是默认为int.
  - 但是不写return的话它还是管不了.

## 7.17 注意事项

### 7.17.1 常见下毒手法

- 0/1base是不是搞混了
- 高精度高低位搞反了吗
- 线性筛抄对了吗
- 快速乘抄对了吗
- $i \leq n, j \leq m$
- sort比较函数是不是比了个寂寞
- 该取模的地方都取模了吗
- 边界情况(+1-1之类的)有没有想清楚
- 特判是否有必要, 确定写对了吗

### 7.17.2 场外相关

- 安顿好之后查一下附近的咖啡店,打印店,便利店之类的位置,以备不时之需
- 热身赛记得检查一下编译注意事项中的代码能否过编译,还有熟悉比赛场地,清楚洗手间在哪儿,测试打印机(如果可以)
- 比赛前至少要翻一遍板子,尤其要看原理与例题
- 比赛前一两天不要摸鱼,要早睡,有条件最好洗个澡;比赛当天不要起太晚,维持好的状态
- 赛前记得买咖啡,最好直接安排三人份,记得要咖啡因比较足的;如果主办方允许,就带些巧克力之类的高热量零食
- 入场之后记得检查机器,尤其要逐个检查键盘按键有没有坏的;如果可以的话,调一下gedit设置
- 开赛之前调整好心态,比赛而已,不必心急.

### 7.17.3 做题策略与心态调节

- 拿到题后立刻按照商量好的顺序读题, 前半小时最好跳过题意太复杂的题(除非被过穿了)
- 签到题写完不要激动, 稍微检查一下最可能的下毒点再交, 避免无谓的罚时
  - 一两行的那种傻逼题就算了
- 读完题及时输出题意, 一方面避免重复读题, 一方面也可以让队友有一个初步印象, 方便之后决定开题顺序
- 如果不能确定题意就不要贸然输出甚至上机, 尤其是签到题, 因为样例一般都很弱
- 一个题如果卡了很久又有其他题可以写, 那不妨先放掉写更容易的题, 不要在一棵树上吊死
  - 不要被一两道题搞得心态爆炸, 一方面急也没有意义, 一方面你很可能真的离AC就差一步
- 榜是不会骗人的, 一个题如果被不少人过了就说明这个题很可能并没有那么难;如果不是有十足的把握就不要轻易开没什么人交的题;另外不要忘记最后一小时会封榜
- 想不出题/找不出毒自然容易犯困, 一定不要放任自己昏昏欲睡, 最好去洗手间冷静一下, 没有条件就站起来踱步
- 思考的时候不要挂机, 一定要在草稿纸上画一画, 最好说出声来最不容易断掉思路
- 出完算法一定要check一下样例和一些trivial的情况, 不然容易写了半天发现写了个假算法
- 上机前有时间就提前给需要思考怎么写的地方打草稿, 不要浪费机时
- 查毒时如果最难的地方反复check也没有问题, 就从头到脚仔仔细细查一遍, 不要放过任何细节, 即使是并查集和sort这种东西也不能想当然
- 后半场如果时间不充裕就不要冒险开难题, 除非真的无事可做
  - 如果是没写过的东西也不要轻举妄动, 在有其他好写的题的时候就等一会再说
- 大多数时候都要听队长安排, 虽然不一定最正确但可以保持组织性
- 任何时候都不要着急, 着急不能解决问题, 不要当蒟国王
- 输了游戏, 还有人生;赢了游戏, 还有人生.

## 7.18 附录: VScode相关

### 7.18.1 插件

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

### 7.18.2 设置选项

- Editor: Insert Spaces (取消勾选, 改为tab缩进)
- Editor: Line Warp (开启折行)
- 改配色, “深色+”: 默认深色”
- 自动保存(F1 → “auto”)
- 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 (Windows可用)

### 7.18.3 快捷键

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

## 7.19 附录: 骂人的艺术——梁实秋

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

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

### 1. 知己知彼

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

### 2. 无骂不如己者

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

3. 适可而止

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

4. 旁敲侧击

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

5. 态度镇定

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

6. 出言典雅

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

7. 以退为进

两人对骂,而自己亦有理屈之处,则处于开骂伊始,特宜注意,最

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

8. 预设埋伏

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

9. 小题大做

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

10. 远交近攻

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

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

7.20 附录: Cheat Sheet

见最后几页。

# Theoretical Computer Science Cheat Sheet

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



# Theoretical Computer Science Cheat Sheet

## Identities Cont.

$$\begin{aligned}
 38. \quad \binom{n+1}{m+1} &= \sum_k \binom{n}{k} \binom{k}{m} = \sum_{k=0}^n \binom{k}{m} n^{\overline{n-k}} = n! \sum_{k=0}^n \frac{1}{k!} \binom{k}{m}, & 39. \quad \begin{bmatrix} x \\ x-n \end{bmatrix} &= \sum_{k=0}^n \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \begin{pmatrix} x+k \\ 2n \end{pmatrix}, \\
 40. \quad \left\{ \begin{matrix} n \\ m \end{matrix} \right\} &= \sum_k \binom{n}{k} \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} (-1)^{n-k}, & 41. \quad \begin{bmatrix} n \\ m \end{bmatrix} &= \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k}, \\
 42. \quad \left\{ \begin{matrix} m+n+1 \\ m \end{matrix} \right\} &= \sum_{k=0}^m k \left\{ \begin{matrix} n+k \\ k \end{matrix} \right\}, & 43. \quad \begin{bmatrix} m+n+1 \\ m \end{bmatrix} &= \sum_{k=0}^m k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix}, \\
 44. \quad \binom{n}{m} &= \sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k}, & 45. \quad (n-m)! \binom{n}{m} &= \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (-1)^{m-k}, \quad \text{for } n \geq m, \\
 46. \quad \left\{ \begin{matrix} n \\ n-m \end{matrix} \right\} &= \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix}, & 47. \quad \begin{bmatrix} n \\ n-m \end{bmatrix} &= \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\}, \\
 48. \quad \left\{ \begin{matrix} n \\ \ell+m \end{matrix} \right\} \binom{\ell+m}{\ell} &= \sum_k \left\{ \begin{matrix} k \\ \ell \end{matrix} \right\} \left\{ \begin{matrix} n-k \\ m \end{matrix} \right\} \binom{n}{k}, & 49. \quad \begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} &= \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}.
 \end{aligned}$$

## Trees

Every tree with  $n$  vertices has  $n-1$  edges.

Kraft inequality: If the depths of the leaves of a binary tree are  $d_1, \dots, d_n$ :

$$\sum_{i=1}^n 2^{-d_i} \leq 1,$$

and equality holds only if every internal node has 2 sons.

## Recurrences

Master method:

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$$

If  $\exists \epsilon > 0$  such that  $f(n) = O(n^{\log_b a - \epsilon})$  then

$$T(n) = \Theta(n^{\log_b a}).$$

If  $f(n) = \Theta(n^{\log_b a})$  then

$$T(n) = \Theta(n^{\log_b a} \log_2 n).$$

If  $\exists \epsilon > 0$  such that  $f(n) = \Omega(n^{\log_b a + \epsilon})$ , and  $\exists c < 1$  such that  $af(n/b) \leq cf(n)$  for large  $n$ , then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that  $T_i$  is always a power of two.

Let  $t_i = \log_2 T_i$ . Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let  $u_i = t_i/2^i$ . Dividing both sides of the previous equation by  $2^{i+1}$  we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \frac{1}{2} + u_i, \quad u_1 = \frac{1}{2},$$

which is simply  $u_i = i/2$ . So we find that  $T_i$  has the closed form  $T_i = 2^{i2^{i-1}}$ .

Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving  $T$  are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side “telescope”

$$1(T(n) - 3T(n/2)) = n$$

$$3(T(n/2) - 3T(n/4)) = n/2$$

$$\vdots \quad \vdots \quad \vdots$$

$$3^{\log_2 n - 1} (T(2) - 3T(1)) = 2$$

Let  $m = \log_2 n$ . Summing the left side we get  $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$  where  $k = \log_2 3 \approx 1.58496$ .

Summing the right side we get

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$$

Let  $c = \frac{3}{2}$ . Then we have

$$n \sum_{i=0}^{m-1} c^i = n \left( \frac{c^m - 1}{c - 1} \right)$$

$$= 2n(c^{\log_2 n} - 1)$$

$$= 2n(c^{(k-1)\log_2 n} - 1)$$

$$= 2n^k - 2n,$$

and so  $T(n) = 3n^k - 2n$ . Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^i T_j.$$

Subtracting we find

$$T_{i+1} - T_i = 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j$$

$$= T_i.$$

And so  $T_{i+1} = 2T_i = 2^{i+1}$ .

Generating functions:

1. Multiply both sides of the equation by  $x^i$ .
2. Sum both sides over all  $i$  for which the equation is valid.
3. Choose a generating function  $G(x)$ . Usually  $G(x) = \sum_{i=0}^{\infty} x^i g_i$ .
3. Rewrite the equation in terms of the generating function  $G(x)$ .
4. Solve for  $G(x)$ .
5. The coefficient of  $x^i$  in  $G(x)$  is  $g_i$ .

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$$

We choose  $G(x) = \sum_{i \geq 0} x^i g_i$ . Rewrite in terms of  $G(x)$ :

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for  $G(x)$ :

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:

$$\begin{aligned}
 G(x) &= x \left( \frac{2}{1-2x} - \frac{1}{1-x} \right) \\
 &= x \left( 2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\
 &= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}.
 \end{aligned}$$

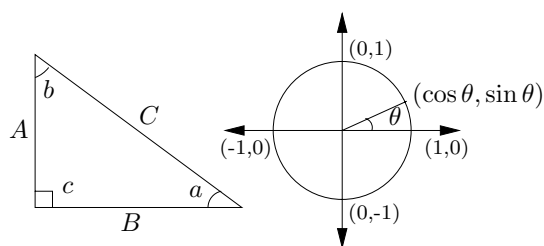
So  $g_i = 2^i - 1$ .



Theoretical Computer Science Cheat Sheet					
$\pi \approx 3.14159,$		$e \approx 2.71828,$	$\gamma \approx 0.57721,$	$\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803,$	$\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$
$i$	$2^i$	$p_i$	General	Probability	
1	2	2	<p>Bernoulli Numbers (<math>B_i = 0</math>, odd <math>i \neq 1</math>): <math>B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},</math> <math>B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.</math></p> <p>Change of base, quadratic formula: <math>\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.</math></p> <p>Euler's number <math>e</math>: <math>e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots</math> <math>\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.</math> <math>\left(1 + \frac{1}{n}\right)^n &lt; e &lt; \left(1 + \frac{1}{n}\right)^{n+1}.</math> <math>\left(1 + \frac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).</math></p> <p>Harmonic numbers: <math>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</math> <math>\ln n &lt; H_n &lt; \ln n + 1,</math> <math>H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).</math></p> <p>Factorial, Stirling's approximation: <math>1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots</math> <math>n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).</math></p> <p>Ackermann's function and inverse: <math display="block">a(i, j) = \begin{cases} 2^j &amp; i = 1 \\ a(i-1, 2) &amp; j = 1 \\ a(i-1, a(i, j-1)) &amp; i, j \geq 2 \end{cases}</math> <math>\alpha(i) = \min\{j \mid a(j, j) \geq i\}.</math></p>	<p>Continuous distributions: If <math>\Pr[a &lt; X &lt; b] = \int_a^b p(x) dx,</math> then <math>p</math> is the probability density function of <math>X</math>. If <math>\Pr[X &lt; a] = P(a),</math> then <math>P</math> is the distribution function of <math>X</math>. If <math>P</math> and <math>p</math> both exist then <math>P(a) = \int_{-\infty}^a p(x) dx.</math></p> <p>Expectation: If <math>X</math> is discrete <math>E[g(X)] = \sum_x g(x) \Pr[X = x].</math></p> <p>If <math>X</math> continuous then <math>E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).</math></p> <p>Variance, standard deviation: <math>\text{VAR}[X] = E[X^2] - E[X]^2,</math> <math>\sigma = \sqrt{\text{VAR}[X]}.</math></p> <p>For events <math>A</math> and <math>B</math>: <math>\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]</math> <math>\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],</math> iff <math>A</math> and <math>B</math> are independent. <math>\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}</math></p> <p>For random variables <math>X</math> and <math>Y</math>: <math>E[X \cdot Y] = E[X] \cdot E[Y],</math> if <math>X</math> and <math>Y</math> are independent. <math>E[X + Y] = E[X] + E[Y],</math> <math>E[cX] = c E[X].</math></p> <p>Bayes' theorem: <math>\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[B A_j] \Pr[A_j]}.</math></p> <p>Inclusion-exclusion: <math>\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] +</math> <math>\sum_{k=2}^n (-1)^{k+1} \sum_{i_1 &lt; \dots &lt; i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].</math></p> <p>Moment inequalities: <math>\Pr\left[ X  \geq \lambda E[X]\right] \leq \frac{1}{\lambda},</math> <math>\Pr\left[ X - E[X]  \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.</math></p> <p>Geometric distribution: <math>\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,</math> <math>E[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.</math></p>	
2	4	3			
3	8	5			
4	16	7			
5	32	11			
6	64	13			
7	128	17			
8	256	19			
9	512	23			
10	1,024	29			
11	2,048	31			
12	4,096	37			
13	8,192	41			
14	16,384	43			
15	32,768	47			
16	65,536	53			
17	131,072	59			
18	262,144	61			
19	524,288	67			
20	1,048,576	71			
21	2,097,152	73			
22	4,194,304	79			
23	8,388,608	83			
24	16,777,216	89			
25	33,554,432	97			
26	67,108,864	101			
27	134,217,728	103			
28	268,435,456	107			
29	536,870,912	109			
30	1,073,741,824	113			
31	2,147,483,648	127			
32	4,294,967,296	131			
Pascal's Triangle			<p>Binomial distribution: <math>\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,</math> <math>E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.</math></p> <p>Poisson distribution: <math>\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.</math></p> <p>Normal (Gaussian) distribution: <math>p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.</math></p> <p>The "coupon collector": We are given a random coupon each day, and there are <math>n</math> different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all <math>n</math> types is <math>nH_n.</math></p>		
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:

$$\sin x = \frac{1}{\csc x}, \quad \cos x = \frac{1}{\sec x},$$

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

$$1 + \tan^2 x = \sec^2 x, \quad 1 + \cot^2 x = \csc^2 x,$$

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

$$\cos x = -\cos(\pi - x), \quad \tan x = \cot\left(\frac{\pi}{2} - x\right),$$

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

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

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

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

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

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

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

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

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

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

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

Euler's equation:

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

## 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.

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

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

$2 \times 2$  and  $3 \times 3$  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:

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

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

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

Identities:

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

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

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

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

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

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

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

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

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

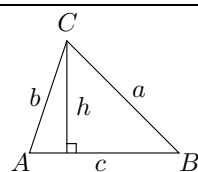
$$2 \sinh^2 \frac{x}{2} = \cosh x - 1, \quad 2 \cosh^2 \frac{x}{2} = \cosh x + 1.$$

$\theta$	$\sin \theta$	$\cos \theta$	$\tan \theta$
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	$\infty$

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

– J. von Neumann

## More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab \cos C.$$

Area:

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

Heron's formula:

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

More identities:

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

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

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

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

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

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

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

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

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

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

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

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

$$\sin x = \frac{\sinh ix}{i},$$

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

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

v2.02 ©1994 by Steve Seiden

sseiden@acm.org

<http://www.csc.lsu.edu/~seiden>

# Theoretical Computer Science Cheat Sheet

## Number Theory

The Chinese remainder theorem: There exists a number  $C$  such that:

$$C \equiv r_1 \pmod{m_1}$$

$$\vdots \quad \vdots \quad \vdots$$

$$C \equiv r_n \pmod{m_n}$$

if  $m_i$  and  $m_j$  are relatively prime for  $i \neq j$ .

Euler's function:  $\phi(x)$  is the number of positive integers less than  $x$  relatively prime to  $x$ . If  $\prod_{i=1}^n p_i^{e_i}$  is the prime factorization of  $x$  then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$$

Euler's theorem: If  $a$  and  $b$  are relatively prime then

$$1 \equiv a^{\phi(b)} \pmod{b}.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \pmod{p}.$$

The Euclidean algorithm: if  $a > b$  are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If  $\prod_{i=1}^n p_i^{e_i}$  is the prime factorization of  $x$  then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers:  $x$  is an even perfect number iff  $x = 2^{n-1}(2^n - 1)$  and  $2^n - 1$  is prime.

Wilson's theorem:  $n$  is a prime iff

$$(n-1)! \equiv -1 \pmod{n}.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n}$$

$$+ O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3}$$

$$+ O\left(\frac{n}{(\ln n)^4}\right).$$

## Graph Theory

### Definitions:

*Loop* An edge connecting a vertex to itself.

*Directed* Each edge has a direction.

*Simple* Graph with no loops or multi-edges.

*Walk* A sequence  $v_0 e_1 v_1 \dots e_\ell v_\ell$ .

*Trail* A walk with distinct edges.

*Path* A trail with distinct vertices.

*Connected* A graph where there exists a path between any two vertices.

*Component* A maximal connected subgraph.

*Tree* A connected acyclic graph.

*Free tree* A tree with no root.

*DAG* Directed acyclic graph.

*Eulerian* Graph with a trail visiting each edge exactly once.

*Hamiltonian* Graph with a cycle visiting each vertex exactly once.

*Cut* A set of edges whose removal increases the number of components.

*Cut-set* A minimal cut.

*Cut edge* A size 1 cut.

*k-Connected* A graph connected with the removal of any  $k-1$  vertices.

*k-Tough*  $\forall S \subseteq V, S \neq \emptyset$  we have  $k \cdot c(G-S) \leq |S|$ .

*k-Regular* A graph where all vertices have degree  $k$ .

*k-Factor* A  $k$ -regular spanning subgraph.

*Matching* A set of edges, no two of which are adjacent.

*Clique* A set of vertices, all of which are adjacent.

*Ind. set* A set of vertices, none of which are adjacent.

*Vertex cover* A set of vertices which cover all edges.

*Planar graph* A graph which can be embedded in the plane.

*Plane graph* An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If  $G$  is planar then  $n - m + f = 2$ , so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree  $\leq 5$ .

### Notation:

$E(G)$  Edge set

$V(G)$  Vertex set

$c(G)$  Number of components

$G[S]$  Induced subgraph

$\deg(v)$  Degree of  $v$

$\Delta(G)$  Maximum degree

$\delta(G)$  Minimum degree

$\chi(G)$  Chromatic number

$\chi_E(G)$  Edge chromatic number

$G^c$  Complement graph

$K_n$  Complete graph

$K_{n_1, n_2}$  Complete bipartite graph

$r(k, \ell)$  Ramsey number

### Geometry

Projective coordinates: triples  $(x, y, z)$ , not all  $x, y$  and  $z$  zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian Projective

$$(x, y) \quad (x, y, 1)$$

$$y = mx + b \quad (m, -1, b)$$

$$x = c \quad (1, 0, -c)$$

Distance formula,  $L_p$  and  $L_\infty$  metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

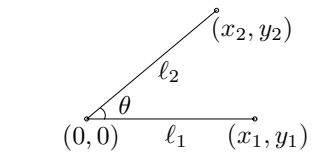
$$[|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p},$$

$$\lim_{p \rightarrow \infty} [|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p}.$$

Area of triangle  $(x_0, y_0)$ ,  $(x_1, y_1)$  and  $(x_2, y_2)$ :

$$\frac{1}{2} \text{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{l_1 l_2}.$$

Line through two points  $(x_0, y_0)$  and  $(x_1, y_1)$ :

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.

– Issac Newton

# Theoretical Computer Science Cheat Sheet

$\pi$

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\frac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \cdots}}}}$$

Gregory's series:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left( 1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$$

Euler's series:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$

$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

## Partial Fractions

Let  $N(x)$  and  $D(x)$  be polynomial functions of  $x$ . We can break down  $N(x)/D(x)$  using partial fraction expansion. First, if the degree of  $N$  is greater than or equal to the degree of  $D$ , divide  $N$  by  $D$ , obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of  $N'$  is less than that of  $D$ . Second, factor  $D(x)$ . Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[ \frac{N(x)}{D(x)} \right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!} \left[ \frac{d^k}{dx^k} \left( \frac{N(x)}{D(x)} \right) \right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.  
– George Bernard Shaw

## Calculus

Derivatives:

$$1. \frac{d(cu)}{dx} = c \frac{du}{dx}, \quad 2. \frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}, \quad 3. \frac{d(uv)}{dx} = u \frac{dv}{dx} + v \frac{du}{dx},$$

$$4. \frac{d(u^n)}{dx} = nu^{n-1} \frac{du}{dx}, \quad 5. \frac{d(u/v)}{dx} = \frac{v \left( \frac{du}{dx} \right) - u \left( \frac{dv}{dx} \right)}{v^2}, \quad 6. \frac{d(e^{cu})}{dx} = ce^{cu} \frac{du}{dx},$$

$$7. \frac{d(c^u)}{dx} = (\ln c) c^u \frac{du}{dx}, \quad 8. \frac{d(\ln u)}{dx} = \frac{1}{u} \frac{du}{dx},$$

$$9. \frac{d(\sin u)}{dx} = \cos u \frac{du}{dx}, \quad 10. \frac{d(\cos u)}{dx} = -\sin u \frac{du}{dx},$$

$$11. \frac{d(\tan u)}{dx} = \sec^2 u \frac{du}{dx}, \quad 12. \frac{d(\cot u)}{dx} = -\csc^2 u \frac{du}{dx},$$

$$13. \frac{d(\sec u)}{dx} = \tan u \sec u \frac{du}{dx}, \quad 14. \frac{d(\csc u)}{dx} = -\cot u \csc u \frac{du}{dx},$$

$$15. \frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}} \frac{du}{dx}, \quad 16. \frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}} \frac{du}{dx},$$

$$17. \frac{d(\arctan u)}{dx} = \frac{1}{1+u^2} \frac{du}{dx}, \quad 18. \frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2} \frac{du}{dx},$$

$$19. \frac{d(\operatorname{arcsec} u)}{dx} = \frac{1}{u\sqrt{1-u^2}} \frac{du}{dx}, \quad 20. \frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx},$$

$$21. \frac{d(\sinh u)}{dx} = \cosh u \frac{du}{dx}, \quad 22. \frac{d(\cosh u)}{dx} = \sinh u \frac{du}{dx},$$

$$23. \frac{d(\tanh u)}{dx} = \operatorname{sech}^2 u \frac{du}{dx}, \quad 24. \frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u \frac{du}{dx},$$

$$25. \frac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u \frac{du}{dx}, \quad 26. \frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u \frac{du}{dx},$$

$$27. \frac{d(\operatorname{arcsinh} u)}{dx} = \frac{1}{\sqrt{1+u^2}} \frac{du}{dx}, \quad 28. \frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}} \frac{du}{dx},$$

$$29. \frac{d(\operatorname{arctanh} u)}{dx} = \frac{1}{1-u^2} \frac{du}{dx}, \quad 30. \frac{d(\operatorname{arcoth} u)}{dx} = \frac{1}{u^2-1} \frac{du}{dx},$$

$$31. \frac{d(\operatorname{arcsech} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx}, \quad 32. \frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{|u|\sqrt{1+u^2}} \frac{du}{dx}.$$

Integrals:

$$1. \int cu \, dx = c \int u \, dx, \quad 2. \int (u+v) \, dx = \int u \, dx + \int v \, dx,$$

$$3. \int x^n \, dx = \frac{1}{n+1} x^{n+1}, \quad n \neq -1, \quad 4. \int \frac{1}{x} \, dx = \ln x, \quad 5. \int e^x \, dx = e^x,$$

$$6. \int \frac{dx}{1+x^2} = \arctan x, \quad 7. \int u \frac{dv}{dx} \, dx = uv - \int v \frac{du}{dx} \, dx,$$

$$8. \int \sin x \, dx = -\cos x, \quad 9. \int \cos x \, dx = \sin x,$$

$$10. \int \tan x \, dx = -\ln |\cos x|, \quad 11. \int \cot x \, dx = \ln |\cos x|,$$

$$12. \int \sec x \, dx = \ln |\sec x + \tan x|, \quad 13. \int \csc x \, dx = \ln |\csc x + \cot x|,$$

$$14. \int \arcsin \frac{x}{a} \, dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0,$$

# 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.  $\int \sinh x dx = \cosh x,$
28.  $\int \cosh x dx = \sinh x,$
29.  $\int \tanh x dx = \ln |\cosh x|,$
30.  $\int \coth x dx = \ln |\sinh x|,$
31.  $\int \operatorname{sech} x dx = \arctan \sinh x,$
32.  $\int \operatorname{csch} x dx = \ln \left| \tanh \frac{x}{2} \right|,$
33.  $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2} x,$
34.  $\int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2} x,$
35.  $\int \operatorname{sech}^2 x dx = \tanh x,$
36.  $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$
37.  $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$
38.  $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
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.  $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|,$
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.

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

## Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$

$$\mathbf{E} f(x) = f(x+1).$$

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:

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

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

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

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

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

Sums:

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

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

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

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

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

Falling Factorial Powers:

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

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

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

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

Rising Factorial Powers:

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

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

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

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

Conversion:

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

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

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

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

$$x^n = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \left[ \begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[ \begin{matrix} n \\ k \end{matrix} \right] x^k.$$

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



# 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:

$$\begin{aligned} \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^n x^2 + 3^n x^3 + 4^n x^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. \end{aligned}$$

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:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$x A'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x) dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

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

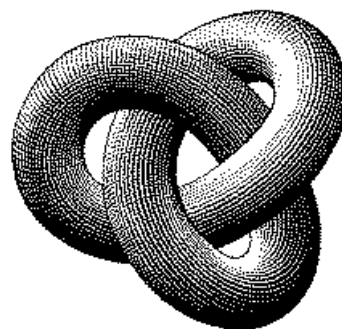
## Series

Expansions:

$$\begin{aligned}\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x} &= \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i, \\ x^{\overline{n}} &= \sum_{i=0}^{\infty} \left[ \begin{matrix} n \\ i \end{matrix} \right] x^i, \\ \left( \ln \frac{1}{1-x} \right)^n &= \sum_{i=0}^{\infty} \left[ \begin{matrix} i \\ n \end{matrix} \right] \frac{n! x^i}{i!}, \\ \tan x &= \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i} (2^{2i} - 1) B_{2i} x^{2i-1}}{(2i)!}, \\ \frac{1}{\zeta(x)} &= \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x}, \\ \zeta(x) &= \prod_p \frac{1}{1 - p^{-x}}, \\ \zeta^2(x) &= \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \sum_{d|n} 1, \\ \zeta(x) \zeta(x-1) &= \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \sum_{d|n} d, \\ \zeta(2n) &= \frac{2^{2n-1} |B_{2n}|}{(2n)!} \pi^{2n}, \quad n \in \mathbb{N}, \\ \frac{x}{\sin x} &= \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2) B_{2i} x^{2i}}{(2i)!}, \\ \left( \frac{1 - \sqrt{1-4x}}{2x} \right)^n &= \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i, \\ e^x \sin x &= \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i, \\ \sqrt{\frac{1 - \sqrt{1-x}}{x}} &= \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2} (2i)! (2i+1)!} x^i, \\ \left( \frac{\arcsin x}{x} \right)^2 &= \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.\end{aligned}$$

$$\begin{aligned}\left( \frac{1}{x} \right)^{\overline{-n}} &= \sum_{i=0}^{\infty} \left\{ \begin{matrix} i \\ n \end{matrix} \right\} x^i, \\ (e^x - 1)^n &= \sum_{i=0}^{\infty} \left\{ \begin{matrix} i \\ n \end{matrix} \right\} \frac{n! x^i}{i!}, \\ x \cot x &= \sum_{i=0}^{\infty} \frac{(-4)^i B_{2i} x^{2i}}{(2i)!}, \\ \zeta(x) &= \sum_{i=1}^{\infty} \frac{1}{i^x}, \\ \frac{\zeta(x-1)}{\zeta(x)} &= \sum_{i=1}^{\infty} \frac{\phi(i)}{i^x},\end{aligned}$$

## Escher's Knot



## Stieltjes Integration

If  $G$  is continuous in the interval  $[a, b]$  and  $F$  is nondecreasing then

$$\int_a^b G(x) dF(x)$$

exists. If  $a \leq b \leq c$  then

$$\int_a^c G(x) dF(x) = \int_a^b G(x) dF(x) + \int_b^c G(x) dF(x).$$

If the integrals involved exist

$$\int_a^b (G(x) + H(x)) dF(x) = \int_a^b G(x) dF(x) + \int_a^b H(x) dF(x),$$

$$\int_a^b G(x) d(F(x) + H(x)) = \int_a^b G(x) dF(x) + \int_a^b G(x) dH(x),$$

$$\int_a^b c \cdot G(x) dF(x) = \int_a^b G(x) d(c \cdot F(x)) = c \int_a^b G(x) dF(x),$$

$$\int_a^b G(x) dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x) dG(x).$$

If the integrals involved exist, and  $F$  possesses a derivative  $F'$  at every point in  $[a, b]$  then

$$\int_a^b G(x) dF(x) = \int_a^b G(x) F'(x) dx.$$

## Cramer's Rule

If we have equations:

$$a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2$$

$$\vdots \quad \quad \quad \vdots$$

$$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$$

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

$$x_i = \frac{\det A_i}{\det A}.$$

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.  
– William Blake (The Marriage of Heaven and Hell)

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system:  
Every integer  $n$  has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where  $k_i \geq k_{i+1} + 2$  for all  $i$ ,  
 $1 \leq i < m$  and  $k_m \geq 2$ .

## Fibonacci Numbers

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Definitions:

$$F_i = F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1,$$

$$F_{-i} = (-1)^{i-1} F_i,$$

$$F_i = \frac{1}{\sqrt{5}} \left( \phi^i - \hat{\phi}^i \right),$$

Cassini's identity: for  $i > 0$ :

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$$

$$F_{2n} = F_n F_{n+1} + F_{n-1} F_n.$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$