

# 目录

<b>I 杂项</b>	<b>6</b>
1 快读快写	6
2 玄学优化	6
3 正则表达式	6
4 随机数	6
5 计算 $\log_2$	7
6 快速开根号   牛顿迭代法	7
7 $i/k == j$ 的 $k$ 的个数	7
8 三分法	8
<b>II 计算几何</b>	<b>8</b>
9 向量坐标直线圆 (结构体)	8
10 二维凸包	11
11 平面最近点对	11
12 最小圆覆盖   随即增量法	12
<b>III 数据结构</b>	<b>13</b>
13 堆	13
14 平衡树	13
14.1 Splay . . . . .	13
15 李超线段树	16
16 吉老师线段树   吉司机线段树	17
17 树套树	17
18 树状数组	17
18.1 一维 . . . . .	17
18.2 二维 . . . . .	17
18.2.1 单点修改区间查询 . . . . .	17
19 可持久化线段树 (可持久化数组)	18
20 可持久化并查集	19

21 可持久化线段树 (主席树)	20
22 分块	21
23 莫队	22
23.1 奇偶性排序	22
23.2 带修改莫队	22
23.3 值域分块	22
23.4 二次离线莫队	22
24 ST 表	24
24.1 一维	24
24.2 二维	24
24.3 反向 ST	25
25 并查集	26
<b>IV 字符串</b>	<b>26</b>
26 回文字符串  manacher 算法	26
26.1 判断 $s[l, r]$ 是否为回文	27
27 KMP	27
28 扩展 KMP Z 函数	28
29 后缀数组  SA	28
29.1 $O(n \log^2 n)$	28
29.2 $O(n)$	29
30 字典树	31
31 AC 自动机	32
<b>V 图论   树论</b>	<b>34</b>
32 树的重心	34
33 最大团	34
34 稳定婚姻匹配	35
35 最小生成树	35
36 二分图	36
36.1 二分图匹配	36
36.2 二分图最小顶点覆盖	36
36.3 最大独立集	36
37 最近公共祖先  LCA	36
37.1 倍增	36
37.2 带权 LCA	37
38 树上差分	38

<b>39 树链剖分</b>	<b>38</b>
<b>40 网络流</b>	<b>40</b>
40.1 最大流 . . . . .	40
40.1.1 EK . . . . .	40
40.1.2 Dinic . . . . .	41
40.2 最小割 . . . . .	42
40.3 费用流 . . . . .	42
40.3.1 MCMF . . . . .	42
40.3.2 ZKW_SPFA . . . . .	43
<b>41 最短路</b>	<b>45</b>
41.1 Floyd . . . . .	45
41.2 Dijkstra . . . . .	45
41.3 SPFA . . . . .	45
<b>42 负环</b>	<b>45</b>
<b>43 割点</b>	<b>46</b>
<b>44 SCC 强连通分量  Tarjan</b>	<b>46</b>
<b>45 缩点</b>	<b>46</b>
<b>46 2-SAT</b>	<b>47</b>
46.1 SCC Tarjan . . . . .	47
46.2 DFS . . . . .	48
<b>47 虚树</b>	<b>49</b>
<b>48 线段树优化建图</b>	<b>50</b>
<b>49 矩阵树定理  Kirchhoff</b>	<b>50</b>
 <b>VI 数论</b>	 <b>51</b>
<b>50 快排</b>	<b>51</b>
<b>51 求逆序对 (归并排序)</b>	<b>51</b>
<b>52 线性基</b>	<b>51</b>
<b>53 矩阵</b>	<b>53</b>
<b>54 高斯消元</b>	<b>55</b>
54.1 异或方程组 . . . . .	55
<b>55 拉格朗日插值</b>	<b>56</b>
<b>56 快速乘</b>	<b>57</b>
<b>57 复数</b>	<b>57</b>
<b>58 快速傅里叶变换  FFT</b>	<b>58</b>
<b>59 快速数论变换  NTT</b>	<b>58</b>

60 任意模数 NTT MTT	59
61 分治 FFT	60
62 快速沃尔什变换  FWT	61
62.1 或运算 . . . . .	61
62.2 与运算 . . . . .	61
62.3 异或运算 . . . . .	61
62.4 code . . . . .	61
63 快速子集变换 (子集卷积) FST	62
63.1 倍增子集卷积 . . . . .	63
64 第二类斯特林数	63
65 约瑟夫环	64
66 最小公倍数 lcm	64
67 扩展欧几里得 (同余方程	64
68 乘法逆元	64
68.1 拓展欧几里得 . . . . .	64
68.2 费马小定理 . . . . .	65
68.3 线性递推 . . . . .	65
69 中国剩余定理	65
69.1 中国剩余定理 CRT(m 互质) . . . . .	65
69.2 扩展中国剩余定理 EXCRT(m 不互质) . . . . .	65
70 排列组合奇偶性	65
71 欧拉函数	66
71.1 筛法 . . . . .	66
72 莫比乌斯函数	66
73 线性筛素数	67
74 判断素数 (质数)	67
74.1 Miller-Rabin 素性测试 . . . . .	67
75 线性筛 GCD	67
76 BSGS	68
77 错排	68
78 原根	68
<b>VII 动态规划 DP</b>	<b>68</b>
79 线性 DP	69
79.1 最长公共子序列 LCS . . . . .	69

<b>80 状压 DP</b>	<b>69</b>
80.1 枚举子集	69
80.2 枚举 $n$ 个元素大小为 $k$ 的二进制子集	69
<b>81 背包问题</b>	<b>69</b>
81.1 多重背包	69
<b>82 斜率优化</b>	<b>70</b>
82.1 「HNOI2008」玩具装箱 TOY	70
<b>83 四边形不等式</b>	<b>70</b>
83.1 2D1D	70
83.2 1D1D	71
83.3 满足四边形不等式的函数类	71
<b>84 插头 DP   轮廓线 DP</b>	<b>71</b>
84.1 一个闭合回路	71
84.2 多个闭合回路	73
84.3 联通块	73
84.4 L 型	74
 <b>VIII STL</b>	 <b>74</b>
<b>85 unordered_map 重载</b>	<b>74</b>

# Part I

## 杂项

### 1 快读快写

```

1 template <typename T> inline void read(T &x) {
2     int c; T tag = 1;
3     while(!isdigit((c=getchar()))) if(c == '-') tag = -1;
4     x = c-'0';
5     while(isdigit((c=getchar()))) x = (x<<1)+(x<<3) + c-'0';
6     x *= tag;
7 }
8 template <typename T> void write(T x) {
9     if(x < 0) x = -x, putchar('-');
10    if(x > 9) write(x/10);
11    putchar(x%10+'0');
12 }

```

```

1 ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);

```

### 2 玄学优化

吸氧，吸臭氧

```

1 #pragma GCC optimize("Ofast,no-stack-protector")
2 #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,avx2,
   tune=native")

```

### 3 正则表达式

```

1 scanf("%3s", str); // 读取长度为n的字符串
2 scanf("%[abc]", str); // 读取a,b,c,读到之外的立即停止
3 scanf("%[a-z0-9]", str); // 同上,读取小写字母和数字
4 scanf("%*[a-z]%s", str); // 过滤掉小写字母读取
5 scanf("%[^a-z]", str); // 读取小写字符外字符,^表示非

```

### 4 随机数

```

1 #include <random> // 范围 unsigned int
2 mt19937 rnd(time(NULL));
3 mt19937 rnd(chrono::high_resolution_clock::now().time_since_epoch().
   count());
4 cout << rnd() << endl;
5
6 std::random_device rd; //获取随机数种子
7 std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded
   with rd()

```

```

8 std::uniform_int_distribution<> dis(0, 9);
9 std::cout << dis(gen) << endl;
10
11 inline ull xorshift128(){
12     static U SX=335634763,SY=873658265,SZ=192849106,SW=746126501;
13     U t=SX^(SX<<11);
14     SX=SY;
15     SY=SZ;
16     SZ=SW;
17     return SW=SW^(SW>>19)^t^(t>>8);
18 }
19 inline ull myrand(){return (xorshift128()<<32)^xorshift128();}

```

## 5 计算 $\log_2$

```

1 #define log(x) (31-__builtin_clz(x))
2 // lg2[i] = lg2(i) + 1
3 for(int i = 1; i <= n; ++i) lg2[i] = lg2[i>>1]+1;
4 // lg2[i] = (int)log2(i)
5 for(int i = 2; i <= n; ++i) lg2[i] = lg2[i>>1]+1;

```

## 6 快速开根号 | 牛顿迭代法

```

1 double sqrt(const double &a) {
2     double x = a, y = .0;
3     while (abs(x-y) > err) {
4         y = x;
5         x = .5*(x+a/x);
6     }
7     return x;
8 }

```

## 7 $i/k == j$ 的 $k$ 的个数

$r-l+1$

```

1 for (int i = 1; i <= n; ++i) {
2     for (int j = 1, l, r; j <= n; ++j) {
3         l = max(1, i/(j+1));
4         while (l-1 >= 1 && i/(l-1) == j) --l;
5         while (i/l > j) ++l;
6         r = i/j;
7         while (r+1 <= i && i/(r+1) == j) ++r;
8         while (i/r < j) --r;
9         if (r-l+1 != i/j-i/(j+1)) cout << i << " " << j << endl;
10    }
11 }

```

## 8 三分法

示例为凹函数

```

1 while (l < r) {
2     int mid = (l+r)>>1;
3     if (f(mid) < f(mid+1)) r = mid;
4     else l = mid+1;
5 }
6 while (r-l > eps) {
7     double ml = l+(r-l)/3, mr = r-(r-l)/3;
8     if (f(ml) < f(mr)) r = mr;
9     else l = ml;
10 }

```

## Part II 计算几何

### 9 向量坐标直线圆 (结构体)

```

1 struct Point {
2     typedef double T;
3     T x, y;
4     int id;
5     Point(){}
6     Point(const T &_x, const T &_y, const int &_i = 0) : x(_x), y(_y),
7         id(_i) {}
8     friend Point operator + (const Point &p1, const Point &p2) {
9         return Point(p1.x+p2.x, p1.y+p2.y, p1.id);
10    }
11    friend Point operator - (const Point &p1, const Point &p2) {
12        return Point(p1.x-p2.x, p1.y-p2.y, p1.id);
13    }
14    friend Point operator - (const Point &p) {
15        return Point(-p.x, -p.y, p.id);
16    }
17    // a*b b在a的顺负逆正
18    friend T operator * (const Point &p1, const Point &p2) {
19        return p1.x*p2.y-p1.y*p2.x;
20    }
21    template <typename TT>
22    friend Point operator / (const Point &p, const TT &k) {
23        return Point(p.x/k, p.y/k, p.id);
24    }
25    template <typename TT>
26    friend Point operator * (const Point &p, const TT &k) {
27        return Point(p.x*k, p.y*k, p.id);
28    }
29    Point operator += (const Point &p) { return *this = *this+p; }
30    Point operator -= (const Point &p) { return *this = *this-p; }
31    template <typename TT>
32    Point operator *= (const TT &k) { return *this = *this*k; }
33    template <typename TT>

```



```

33 Point operator /= (const TT &k) { return *this = *this/k; }
34 friend bool operator < (const Point &p1, const Point &p2) {
35     return make_pair(p1.x, p1.y) < make_pair(p2.x, p2.y);
36 }
37 friend bool operator > (const Point &p1, const Point &p2) {
38     return make_pair(p1.x, p1.y) > make_pair(p2.x, p2.y);
39 }
40 friend bool operator == (const Point &p1, const Point &p2) {
41     return p1.x == p2.x && p1.y == p2.y;
42 }
43 friend bool operator != (const Point &p1, const Point &p2) {
44     return p1.x != p2.x || p1.y != p2.y;
45 }
46 friend istream& operator >> (istream &is, Point &p) {
47     return is >> p.x >> p.y;
48 }
49 friend ostream& operator << (ostream &os, Point &p) {
50     return os << p.x << " " << p.y << " " << p.id << endl;
51 }
52 double length() { return sqrt(1.0*x*x+1.0*y*y); }
53 friend double dis(const Point &p1, const Point &p2) { return (p2-p1
54     ).length(); }
54 double dis(const Point &p) { return (p-*this).length(); }
55 friend T dot(const Point &p1, const Point &p2) { return p1.x*p2.x+
56     p1.y*p2.y; }
56 T dot(const Point &p) { return x*p.x+y*p.y; }
57 friend Point rotate_90_c(const Point &p) { return Point(p.y, -p.x,
58     p.id); }
58 Point rotate_90_c() { return Point(y, -x, id); }
59 friend double atan(const Point &p) { return atan2(p.y, p.x); }
60 };
61
62 template <typename T = double>
63 struct Vec { // 三维向量
64     T x, y, z;
65     Vec(const T &_x = 0, const T &_y = 0, const T &_z = 0) : x(_x), y(
66         _y), z(_z) {}
66     double len() { return sqrt(1.0*x*x+1.0*y*y+1.0*z*z); }
67     friend Vec operator +(const Vec &v1, const Vec &v2) { return Vec(v1
68         .x+v2.x, v1.y+v2.y, v1.z+v2.z); }
68     friend Vec operator -(const Vec &v1, const Vec &v2) { return Vec(v1
69         .x-v2.x, v1.y-v2.y, v1.z-v2.z); }
69     friend Vec operator *(const T &k, const Vec &v) { return Vec(k*v.x,
70         k*v.y, k*v.z); }
70     friend Vec operator *(const Vec &v, const T &k) { return k*v; }
71     friend Vec operator *(const Vec &v1, const Vec &v2) {
72         return Vec(
73             v1.y*v2.z-v1.z*v2.y,
74             v1.z*v2.x-v1.x*v2.z,
75             v1.x*v2.y-v1.y*v2.x
76         );
77     }
78     friend T dot(const Vec &v1, const Vec &v2) { return v1.x*v2.x+v1.y*
79         v2.y+v1.z*v2.z; }
79     T dot(const Vec &v) { return dot(*this, v); }
80     Vec& operator +=(const Vec &v) { return *this = *this+v; }
81     Vec& operator -=(const Vec &v) { return *this = *this-v; }

```

```

82 Vec& operator *=(const T &k) { return *this = *this*k; }
83 Vec& operator *=(const Vec &v) { return *this = *this*v; }
84 friend istream& operator >>(istream &is, Vec &v) { return is >> v.x
    >> v.y >> v.z; }
85 };
86
87 inline bool polar_angle1(const Point &p1, const Point &p2) {
88     double d1 = atan(p1), d2 = atan(p2);
89     return d1 == d2 ? p1 < p2 : d1 < d2;
90 }
91
92 inline bool polar_angle2(const Point &p1, const Point &p2) {
93     auto tmp = p1*p2;
94     return tmp == 0 ? p1 < p2 : tmp > 0;
95 }
96
97 inline long long S(const Point &p1, const Point &p2, const Point &p3)
98 {
99     return abs(p1.x*p2.y+p1.y*p3.x+p2.x*p3.y-p1.x*p3.y-p1.y*p2.x-p2.y*
100 p3.x);
101 }
102
103 struct Line {
104     Point p1, p2;
105     Line(){}
106     Line(const Point &p1, const Point &p2) : p1(p1), p2(p2) {}
107     friend bool cross(const Line &l1, const Line &l2) {
108         #define SJ1(x) max(l1.p1.x, l1.p2.x) < min(l2.p1.x, l2.p2.x) || \
109             max(l2.p1.x, l2.p2.x) < min(l1.p1.x, l1.p2.x)
110         if (SJ1(x) || SJ1(y)) return false;
111         #undef SJ1
112         #define SJ2(a, b, c, d) ((a-b)*(a-c))*((a-b)*(a-d)) <= 0
113         return SJ2(l1.p1, l1.p2, l2.p1, l2.p2) &&
114             SJ2(l2.p1, l2.p2, l1.p1, l1.p2);
115         #undef SJ2
116     }
117     friend bool on_line(const Line &l, const Point &p) {
118         return abs((l.p1-l.p2)*(l.p1-p)) < err;
119     }
120     friend Point cross_point(const Line &l1, const Line &l2) {
121         Point v1 = l1.p2-l1.p1, v2 = l2.p2-l2.p1;
122         if (abs(v1*v2) < err) return Point(0, 0); // no cross_point
123         double t = (l2.p1-l1.p1)*v2/(v1*v2);
124         return l1.p1+v1*t;
125     }
126 };
127
128 struct Circular {
129     Point o;
130     double r;
131     Circular(){}
132     Circular(const Point &o, const double &r) : o(o), r(r) {}
133     template <typename T>
134     Circular(const T &x, const T &y, const double &r) : o(Point(x,
135         y)), r(r) {}
136     friend bool in_cir(const Circular &c, const Point &p) { return dis(
137         c.o, p) <= c.r; }

```

```

134 |   bool in_cir(const Point &p) { return dis(o, p) <= r; }
135 | };
136 |
137 | inline Circular get_cir(const Point &p1, const Point &p2, const Point
    |   &p3) {
138 |     Circular res;
139 |     res.o = cross_point(Line((p1+p2)/2, (p1+p2)/2+(p2-p1).rotate_90_c()
    |       ),
140 |       Line((p1+p3)/2, (p1+p3)/2+(p3-p1).rotate_90_c()));
141 |     res.r = dis(res.o, p1);
142 |     return res;
143 | }

```

## 10 二维凸包

```

1 | int n;
2 | int stk[N], used[N], tp;
3 | Point p[N];
4 |
5 | inline void Andrew() {
6 |     memset(used, 0, sizeof used);
7 |     sort(p+1, p+n+1);
8 |     tp = 0;
9 |     stk[++tp] = 1;
10 |    for (int i = 2; i <= n; ++i) {
11 |        while (tp >= 2 && (p[stk[tp]]-p[stk[tp-1]])*(p[i]-p[stk[tp]]) <=
    |            0)
12 |            used[stk[tp--]] = 0;
13 |        used[i] = 1;
14 |        stk[++tp] = i;
15 |    }
16 |    int tmp = tp;
17 |    for (int i = n-1; i; --i) {
18 |        if (used[i]) continue;
19 |        while (tp >= tmp && (p[stk[tp]]-p[stk[tp-1]])*(p[i]-p[stk[tp]])
    |            <= 0)
20 |            used[stk[tp--]] = 0;
21 |        used[i] = 1;
22 |        stk[++tp] = i;
23 |    }
24 | }

```

## 11 平面最近点对

```

1 | Point a[N];
2 | int n, ansa, ansb;
3 | double mindist;
4 |
5 | inline bool cmp_y(const Point &p1, const Point &p2) { return p1.y <
    |     p2.y; }
6 |
7 | void upd_ans(const Point &p1, const Point &p2) {

```

```

8   double dist = dis(p1, p2);
9   if (dist < mindist) mindist = dist, ansa = p1.id, ansb = p2.id;
10  }
11
12  void rec(int l, int r) {
13      if (r-l <= 3) {
14          for (int i = l; i < r; ++i)
15              for (int j = i+1; j <= r; ++j)
16                  upd_ans(a[i], a[j]);
17          sort(a+l, a+r+1, cmp_y);
18          return;
19      }
20
21      static Point t[N];
22      int m = (l+r)>>1, midx = a[m].x;
23      rec(l, m); rec(m+1, r);
24      merge(a+l, a+m+1, a+m+1, a+r+1, t, cmp_y);
25      copy(t, t+r-l+1, a+l);
26
27      int tsz = 0;
28      for (int i = l; i <= r; ++i)
29          if (abs(a[i].x-midx) <= mindist) {
30              for (int j = tsz; j && a[i].y-t[j].y < mindist; --j)
31                  upd_ans(a[i], t[j]);
32              t[++tsz] = a[i];
33          }
34  }
35
36  inline void mindist_pair() {
37      sort(a+1, a+n+1);
38      mindist = INF;
39      rec(1, n);
40  }

```

## 12 最小圆覆盖 | 随即增量法

```

1  inline Circular RIA() {
2      Circular cir;
3      random_shuffle(a+1, a+n+1);
4      for (int i = 1; i <= n; ++i) {
5          if (cir.in_cir(a[i])) continue;
6          cir = Circular(a[i], 0);
7          for (int j = 1; j < i; ++j) {
8              if (cir.in_cir(a[j])) continue;
9              cir = Circular((a[i]+a[j])/2, dis(a[i], a[j])/2);
10             for (int k = 1; k < j; ++k) {
11                 if (cir.in_cir(a[k])) continue;
12                 cir = get_cir(a[i], a[j], a[k]);
13             }
14         }
15     }
16     return cir;
17 }

```

## Part III

# 数据结构

## 13 堆

```

1 struct Heap {
2     static const int Maxn = 1e6+7;
3     int sz, a[Maxn];
4     Heap() { sz = 0; memset(a, 0, sizeof a); }
5     inline bool cmp(int x, int y) { return x < y; } // 小根堆
6     inline int size() { return sz; }
7     inline bool empty() { return sz == 0; }
8     inline int top() { return a[1]; }
9     inline void push(int x) { a[++sz] = x; swift_up(sz); }
10    inline void pop() { swap(a[1], a[sz--]); swift_down(1); }
11    inline void swift_up(int p) {
12        while(p > 1 && cmp(a[p], a[p>>1])) // a[p] < a[p<<1]
13            swap(a[p], a[p>>1]), p >>= 1;
14    }
15    inline void swift_down(int p) {
16        int l, r, s;
17        while(true) {
18            l = p<<1; r = p<<1|1;
19            if(l > sz) break;
20            if(r > sz || cmp(a[l], a[r])) s = l; // a[l] < a[r]
21            else s = r;
22            if(cmp(a[s], a[p])) // a[s] < a[p]
23                swap(a[p], a[s]), p = s;
24            else break;
25        }
26    }
27 };

```

## 14 平衡树

### 14.1 Splay

```

1 struct Splay {
2     #define root e[0].ch[1]
3     typedef int T;
4     struct node {
5         T v = 0;
6         int ch[2] = { 0, 0 };
7         int fa = 0, sum = 0, cnt = 0;
8     } e[N];
9     int n;
10    void update(int x) { e[x].sum = e[e[x].ch[0]].sum+e[e[x].ch[1]].sum
        +e[x].cnt; }
11    int identify(int x) { return x == e[e[x].fa].ch[1]; }
12    void connect(int x,int f,int son) { e[x].fa = f; e[f].ch[son] = x;
        }
13    void rotate(int x) {

```

```

14     int y = e[x].fa,
15         r = e[y].fa,
16         rson = identify(y),
17         yson = identify(x),
18         b = e[x].ch[yson^1];
19     connect(b, y, yson);
20     connect(y, x, yson^1);
21     connect(x, r, rson);
22     update(y); update(x);
23 }
24 void splay(int at, int to) {
25     to = e[to].fa;
26     int up;
27     while((up = e[at].fa) != to) {
28         if(e[up].fa != to)
29             rotate(identify(up) == identify(at) ? up : at);
30         rotate(at);
31     }
32 }
33 int add_point(T v, int fa) {
34     ++n; e[n].v = v; e[n].fa = fa; e[n].sum = e[n].cnt = 1;
35     return n;
36 }
37 int find(T v) {
38     int now = root, last = 0;
39     while (now && e[now].v != v)
40         last = now, now = e[now].ch[v > e[now].v];
41     splay((now ? now : last), root);
42     return now;
43 }
44 void insert(T v) {
45     if (!root) { root = add_point(v, root); return; }
46     int now = root, last = 0;
47     while (now && e[now].v != v)
48         last = now, now = e[now].ch[v > e[now].v];
49     if (now) ++e[now].cnt;
50     else now = e[last].ch[v > e[last].v] = add_point(v, last);
51     splay(now, root);
52 }
53 void erase(T v) {
54     int del = find(v);
55     if (!del) return;
56     if (e[del].cnt > 1) {
57         --e[del].cnt;
58         --e[del].sum;
59     } else if (!e[del].ch[0]) {
60         root = e[del].ch[1];
61         e[root].fa = 0;
62     } else {
63         int oldroot = root;
64         splay(nex(e[del].ch[0], 1), root);
65         connect(e[oldroot].ch[1], root, 1);
66         update(root);
67     }
68 }
69 int rank(T v) { return e[e[find(v)].ch[0]].sum+1; }
70 T atrank(int x) {

```

```

71     if (x > e[root].sum) return -INF;
72     int now = root;
73     while (true) {
74         if (x <= e[e[now].ch[0]].sum) now = e[now].ch[0];
75         else if ((x -= e[e[now].ch[0]].sum) <= e[now].cnt) break;
76         else x -= e[now].cnt, now = e[now].ch[1];
77     }
78     splay(now, root);
79     return e[now].v;
80 }
81 // small 0, big 1
82 int nex(int x, int opt) { while (e[x].ch[opt]) x = e[x].ch[opt];
    return x; }
83 T lower(T v, int opt) {
84     insert(v);
85     T res = e[nex(e[root].ch[opt], opt^1)].v;
86     erase(v);
87     return res;
88 }
89 #undef root
90 };

```

## 区间反转

```

1  struct Splay {
2      typedef int T;
3      struct node {
4          T v = 0;
5          int ch[2] = { 0, 0 };
6          int fa = 0, sum = 0, cnt = 0, tag = 0;
7      } e[N];
8      int sz, &root = e[0].ch[1];
9      void update(int x) { e[x].sum = e[e[x].ch[0]].sum+e[e[x].ch[1]].sum
    +e[x].cnt; }
10     int identify(int x) { return x == e[e[x].fa].ch[1]; }
11     void connect(int x,int f,int son) { e[x].fa = f; e[f].ch[son] = x;
    }
12     void rotate(int x) {
13         int y = e[x].fa,
14             r = e[y].fa,
15             rson = identify(y),
16             yson = identify(x),
17             b = e[x].ch[yson^1];
18         connect(b, y, yson);
19         connect(y, x, yson^1);
20         connect(x, r, rson);
21         update(y); update(x);
22     }
23     void splay(int at,int to = 0) {
24         to = e[to].fa;
25         int up;
26         while((up = e[at].fa) != to) {
27             if(e[up].fa != to)
28                 rotate(identify(up) == identify(at) ? up : at);
29             rotate(at);
30         }
31     }
32     int add_point(T v, int fa) {

```

```

33     ++sz; e[sz].v = v; e[sz].fa = fa; e[sz].sum = e[sz].cnt = 1;
34     return sz;
35 }
36 int find(int x) {
37     if (x > e[root].sum) return -INF;
38     int now = root;
39     while (true) {
40         push_down(now);
41         if (x <= e[e[now].ch[0]].sum) now = e[now].ch[0];
42         else if ((x -= e[e[now].ch[0]].sum) <= e[now].cnt) break;
43         else x -= e[now].cnt, now = e[now].ch[1];
44     }
45     return now;
46 }
47 int build(int l, int r, int fa) {
48     if (l > r) return 0;
49     int mid = (l+r)>>1,
50     now = add_point(mid, fa);
51     e[now].ch[0] = build(l, mid-1, now);
52     e[now].ch[1] = build(mid+1, r, now);
53     update(now);
54     return now;
55 }
56 void push_down(int x) {
57     if (x && e[x].tag) {
58         e[e[x].ch[0]].tag ^= 1;
59         e[e[x].ch[1]].tag ^= 1;
60         swap(e[x].ch[0], e[x].ch[1]);
61         e[x].tag = 0;
62     }
63 }
64 void reverse(int l, int r) {
65     int pl = find(l-1+1), pr = find(r+1+1);
66     splay(pl); splay(pr, pl);
67     e[e[e[root].ch[1]].ch[0]].tag ^= 1;
68 }
69 void print_LMR(int x) {
70     if (!x) return;
71     push_down(x);
72     print_LMR(e[x].ch[0]);
73     if (e[x].v != 0 && e[x].v != n+1)
74         write(a[e[x].v]), putchar(' ');
75     print_LMR(e[x].ch[1]);
76 }
77 } tree;

```

## 15 李超线段树

李超线段树是一种用于维护平面直角坐标系内线段关系的数据结构。它常被用来处理这样一种形式的问题：给定一个平面直角坐标系，支持动态插入一条线段，询问从某一个位置  $(x, +\infty)$  向下看能看到的最高的一条线段（也就是给一条竖线，问这条竖线与所有线段的最高的交点）。



## 16 吉老师线段树 | 吉司机线段树

区间最值操作 & 区间历史最值

栗子：给出一个数列，每次操作让某个区间中对给定值取  $\min$  询问某个区间的和

## 17 树套树

在第一维线段树的每个结点建立第二维线段树

## 18 树状数组

### 18.1 一维

单点修改区间查询

区间修改单点查询

```

1 template <typename T>
2 struct BinaryIndexedTree {
3     int n;
4     T tr[N];
5     BinaryIndexedTree() { memset(tr, 0, sizeof tr); }
6     void init(const int &n) { n = _n; clear(); }
7     void clear() { memset(tr+1, 0, sizeof(T)*n); }
8     void add(const int &x, const T &v) { for (int i = x; i <= n; i += i
9         &-i) tr[i] += v; }
10    void add(const int &x, const int &y, const T &v) { add(x, v); add(y
11        +1, -v); }
12    T query(const int &x) { T res = 0; for (int i = x; i; i -= i&-i)
13        res += tr[i]; return res; }
14    T query(const int &x, const int &y) { return query(y)-query(x-1); }
15 };

```

$O(n)$  初始化

```

1 template <typename TT>
2 void init(const int &n, const TT a[]) {
3     n = _n; clear();
4     for (int i = 1; i <= n; ++i) {
5         tr[i] += a[i];
6         if (i+(i&-i) <= n) tr[i+(i&-i)] += tr[i];
7     }
8 }

```

### 18.2 二维

#### 18.2.1 单点修改区间查询

```

1 template <typename T>
2 struct BIT_2D {
3     int n, m;
4     T a[N][N], tr[N][N];
5     BIT_2D() { memset(tr, 0, sizeof tr); }
6     void init(const int &n, const int &m) {
7         n = _n; m = _m;

```

```

8     memset(a, 0, sizeof a);
9     memset(tr, 0, sizeof tr);
10 }
11 void add(const int &x, const int &y, const T &k) {
12     a[x][y] += k;
13     for (int i = x; i <= n; i += i&-i)
14         for (int j = y; j <= m; j += j&-j)
15             tr[i][j] += k;
16 }
17 T query(const int &x, const int &y) {
18     return a[x][y];
19     // return query(x, y, x, y);
20 }
21 T query(int r1, int c1, int r2, int c2) {
22     if (r1 > r2) swap(r1, r2);
23     if (c1 > c2) swap(c1, c2);
24     return _query(r2, c2) - _query(r1-1, c2) - _query(r2, c1-1) + _query(r1-1, c1-1);
25 }
26 T _query(const int &x, const int &y) {
27     T res = 0;
28     for (int i = x; i; i -= i&-i)
29         for (int j = y; j; j -= j&-j)
30             res += tr[i][j];
31     return res;
32 }
33 };

```

## 19 可持久化线段树 (可持久化数组)

```

1 template <typename T>
2 struct PersistentArray {
3     static const int NN = N*(log2(N)+3);
4     int rt[N], ls[NN], rs[NN], val[NN], tot, n;
5     void build(const int &n) {
6         this->n = n;
7         tot = 0;
8         rt[0] = build(1, n);
9     }
10    int build(const int &l, const int &r) {
11        int cur = ++tot; assert(tot < NN);
12        if (l == r) return val[cur] = a[l], cur;
13        int mid = (l+r)>>1;
14        ls[cur] = build(l, mid);
15        rs[cur] = build(mid+1, r);
16        return cur;
17    }
18    void update(const int &cur, const int &pre, const int &x, const T &k) {
19        rt[cur] = update(rt[pre], x, k, 1, n);
20    }
21    int update(const int &pre, const int &x, const T &k, const int &l,
22              const int &r) {
23        int cur = ++tot; assert(tot < NN);
24        if (l == x && r == x) return val[cur] = k, cur;

```

```

24     ls[cur] = ls[pre]; rs[cur] = rs[pre];
25     int mid = (l+r)>>1;
26     if (x <= mid) ls[cur] = update(ls[pre], x, k, l, mid);
27     else rs[cur] = update(rs[pre], x, k, mid+1, r);
28     return cur;
29 }
30 T query(const int &cur, const int &x) {
31     return query(rt[cur], x, 1, n);
32 }
33 T query(const int &cur, const int &x, const int &l, const int &r) {
34     if (l == x && r == x) return val[cur];
35     int mid = (l+r)>>1;
36     if (x <= mid) return query(ls[cur], x, l, mid);
37     return query(rs[cur], x, mid+1, r);
38 }
39 };

```

## 20 可持久化并查集

```

1 struct PersistentUnionSet {
2     static const int NN = N*(log2(N)+3);
3     int rt[N], ls[NN], rs[NN], fa[NN], dep[NN], n, tot;
4     void build(const int &n) {
5         this->n = n;
6         tot = 0;
7         rt[0] = build(1, n);
8     }
9     int build(const int &l, const int &r) {
10         int cur = ++tot; assert(tot < NN);
11         if (l == r) return fa[cur] = l, dep[cur] = 0, cur;
12         int mid = (l+r)>>1;
13         ls[cur] = build(l, mid);
14         rs[cur] = build(mid+1, r);
15         return cur;
16     }
17     bool query(const int &cur, const int &x, const int &y) {
18         return fa[getf(rt[cur], x)] == fa[getf(rt[cur], y)];
19     }
20     // return the id of fa[], dep[]
21     int query(const int &cur, const int &x, const int &l, const int &r)
22     {
23         if (l == r) return cur;
24         int mid = (l+r)>>1;
25         if (x <= mid) return query(ls[cur], x, l, mid);
26         else return query(rs[cur], x, mid+1, r);
27     }
28     // return the id of fa[], dep[]
29     int getf(const int &cur, int x) {
30         int fi;
31         while (fa[(fi = query(cur, x, 1, n))] != x) x = fa[fi];
32         return fi;
33     }
34     void merge(const int &cur, const int &pre, const int &x, const int
35         &y) {
36         rt[cur] = rt[pre];

```

```

35     int fx = getf(rt[cur], x), fy = getf(rt[cur], y);
36     if (fa[fx] == fa[fy]) return;
37     if (dep[fx] > dep[fy]) swap(fx, fy);
38     rt[cur] = update(rt[pre], fa[fx], fa[fy], 1, n);
39     if (dep[fx] == dep[fy]) add(rt[cur], fa[fy], 1, n);
40 }
41 // update fa, merge x to y
42 int update(const int &pre, const int &x, const int &y, const int &l
    , const int &r) {
43     int cur = ++tot; assert(tot < NN);
44     if (l == r) return fa[cur] = y, dep[cur] = dep[pre], cur;
45     ls[cur] = ls[pre]; rs[cur] = rs[pre];
46     int mid = (l+r)>>1;
47     if (x <= mid) ls[cur] = update(ls[pre], x, y, l, mid);
48     else rs[cur] = update(rs[pre], x, y, mid+1, r);
49     return cur;
50 }
51 // add dep
52 void add(const int &cur, const int &x, const int &l, const int &r)
    {
53     if (l == r) return ++dep[cur], void();
54     int mid = (l+r)>>1;
55     if (x <= mid) add(ls[cur], x, l, mid);
56     else add(rs[cur], x, mid+1, r);
57 }
58 };

```

## 21 可持久化线段树 (主席树)

```

1  template <typename T>
2  struct PersistentSegmentTree {
3      static const int NN = N*(log2(N)+5);
4      int rt[N], sum[NN], ls[NN], rs[NN], tot, n;
5      void build(const int &n) {
6          this->n = n;
7          tot = 0;
8          rt[0] = _build(1, n);
9      }
10     void update(const int &cur, const int &pre, const T &k) {
11         rt[cur] = _update(rt[pre], 1, n, k);
12     }
13     T query(const int &l, const int &r, const int &k) {
14         return _query(rt[l-1], rt[r], 1, n, k);
15     }
16 private:
17     int _build(const int &l, const int &r) {
18         int cur = ++tot;
19         sum[cur] = 0;
20         if (l >= r) return cur;
21         int mid = (l+r)>>1;
22         ls[cur] = _build(l, mid);
23         rs[cur] = _build(mid+1, r);
24         return cur;
25     }

```

```

26 int _update(const int &pre, const int &l, const int &r, const int &
    k) {
27     int cur = ++tot;
28     ls[cur] = ls[pre]; rs[cur] = rs[pre]; sum[cur] = sum[pre]+1;
29     if (l >= r) return cur;
30     int mid = (l+r)>>1;
31     if (k <= mid) ls[cur] = _update(ls[pre], l, mid, k);
32     else rs[cur] = _update(rs[pre], mid+1, r, k);
33     return cur;
34 }
35 int _query(const int &u, const int &v, const int &l, const int &r,
    const int &k) {
36     if (l >= r) return l;
37     int num = sum[ls[v]]-sum[ls[u]], mid = (l+r)>>1;
38     if (num >= k) return _query(ls[u], ls[v], l, mid, k);
39     else return _query(rs[u], rs[v], mid+1, r, k-num);
40 }
41 };

```

## 22 分块

```

1 struct FenKuai {
2     typedef int T;
3     int t; // 每组大小
4     T a[N], b[N], add[N];
5     FenKuai() {
6         memset(a, 0, sizeof a);
7         memset(b, 0, sizeof b);
8         memset(add, 0, sizeof add);
9     }
10    void build(int x) {
11        for (int i = x*t; i < min(x*t+t, n); ++i) b[i] = a[i];
12        sort(b+x*t, b+min(x*t+t, n));
13    }
14    void init() {
15        t = static_cast<int>(sqrt(n)+0.5);
16        for (int i = 0; i*t < n; ++i) build(i);
17    }
18    void update(int x, int y, T c) {
19        int i = x;
20        for ( ; i <= y && i%t; ++i) a[i] += c;
21        build(x/t);
22        for ( ; i+t-1 <= y; i += t) add[i/t] += c;
23        for ( ; i <= y; ++i) a[i] += c;
24        build(y/t);
25    }
26    T query(int x, int y, long long c) {
27        T res = 0; int i = x;
28        for ( ; i <= y && i%t; ++i) res += (a[i]+add[i/t] < c*c);
29        for ( ; i+t-1 <= y; i += t) res += lower_bound(b+i, b+i+t, c*c-
            add[i/t])-(b+i);
30        for ( ; i <= y; ++i) res += (a[i]+add[i/t] < c*c);
31        return res;
32    }
33 } B;

```

## 23 莫队

$O(1)$  一般取  $block = \frac{n}{\sqrt{m}}, O(n\sqrt{m})$

移动前两步先扩大区间  $l--, r++$  后两步缩小区间  $l++, r--$

### 23.1 奇偶性排序

```
1 template <typename T> bool cmp(const T &q1, const T &q2) {
2     return q1.l/block != q2.l/block ? q1.l < q2.l :
3         (q1.l/block)&1 ? q1.r < q2.r : q1.r > q2.r;
4 }
```

### 23.2 带修改莫队

以  $n^{\frac{2}{3}}$  为一块，分成了  $n^{\frac{1}{3}}$  块，第一关键字是左端点所在块，第二关键字是右端点所在块，第三关键字是时间。复杂度  $O(n^{\frac{5}{3}})$

```
1 template <typename T> bool cmp(const T &q1, const T &q2) {
2     return q1.l/block != q2.l/block ? q1.l < q2.l :
3         q1.r/block != q2.r/block ? q1.r < q2.r : q1.t < q2.t;
4 }
```

### 23.3 值域分块

维护块的前缀和以及块内部前缀和， $O(\sqrt{n})$  修改， $O(1)$  求区间和

```
1 template <typename T> struct PreSum {
2     int n, block;
3     T s[N], t[(int)sqrt(N)+3];
4     void init(int n) { this->n = n; block = sqrt(n); }
5     void add(int x, T k) {
6         for (int i = x; i/block == x/block && i <= n; ++i) s[i] += k;
7         for (int i = x/block+1; i <= n/block; ++i) t[i] += k;
8     }
9     T query(int x) { return t[x/block]+s[x]; }
10 };
```

### 23.4 二次离线莫队

大概是一种需要维护信息具有可减性的莫队。只要具有可减性，就可以容斥，就可以二次离线。所谓『二次离线』，大概是指由于普通莫队无法快速计算贡献，所以第一次离线把询问离线下来，第二次离线把莫队的转移过程离线下来。

由于信息具有可减性（比如常见的「点对数」），记  $(a,b)(c,d)$  表示区间  $[a,b]$  内的点和区间  $[c,d]$  内的点对彼此产生的贡献（区间内部不算）。

$$[l, r] \rightarrow [l+t, r], \sum_{i=l}^{l+t-1} (i, i)(i+1, r) = \sum_{i=l}^{l+t-1} (i, i)(1, r) - (i, i)(1, i)$$

$$[l, r] \rightarrow [l-t, r], \sum_{i=l-t}^{l-1} (i, i)(i+1, r) = \sum_{i=l-t}^{l-1} (i, i)(1, r) - (i, i)(1, i)$$

$$[l, r] \rightarrow [l, r+t], \sum_{i=r+1}^{r+t} (i, i)(l, i-1) = \sum_{i=r+1}^{r+t} (1, i-1)(i, i) - (1, l-1)(i, i)$$

$$[l, r] \rightarrow [l, r-t], \sum_{i=r-t+1}^r (i, i)(l, i-1) = \sum_{i=r-t+1}^r (1, i-1)(i, i) - (1, l-1)(i, i)$$

对于  $(1, i-1)(i, i)$  没什么好说，暴力处理前缀和

对于  $(1, l-1)(i, i)$  由于莫队的复杂度，至多有  $n\sqrt{m}$  个不同询问，把每个询问打标记到左端点（比如  $[l, r] \rightarrow [l, r-t]$  就打到  $l-1$  上），最后扫一遍全部  $i \in [1, n]$ ，处理出询问值，因为此时  $i$  枚举  $O(n)$  次，可以用『值域分块』技巧。这样最终复杂度  $O(n\sqrt{n} + n\sqrt{n})$

```

1 Query q[N];
2 SufSum<int> suml;
3 PreSum<int> sumr;
4 vector<Query> ql[N], qr[N];
5
6 inline void calc_sumi() {
7     static BinaryIndexedTree<int> tree;
8     tree.init(n);
9     for (int i = 1; i <= n; ++i) {
10         suml[i] = suml[i-1] + i - tree.query(a[i]);
11         tree.add(a[i], 1);
12     }
13     tree.clear();
14     for (int i = n; i; --i) {
15         sumr[i] = sumr[i+1] + tree.query(a[i]-1);
16         tree.add(a[i], 1);
17     }
18 }
19
20 signed main() {
21     sort(q+1, q+m+1, cmp);
22     calc_sumi();
23     q[0] = Query(0, 1, 0);
24     for (int i = 1, ul, vl, ur, vr; i <= m; ++i) {
25         ul = q[i-1].l; ur = q[i-1].r;
26         vl = q[i].l; vr = q[i].r;
27         res[i] = suml[vr] - suml[ur] + sumr[vl] - sumr[ul];
28         if (vl < ul) qr[vr+1].emplace_back(-i, vl, ul-1);
29         if (vl > ul) qr[vr+1].emplace_back(+i, ul, vl-1);
30         if (vr < ur) ql[ul-1].emplace_back(+i, vr+1, ur);
31         if (vr > ur) ql[ul-1].emplace_back(-i, ur+1, vr);
32     }
33     suml.init(n+1);
34     for (int i = 1; i <= n; ++i) {
35         suml.add(a[i], 1);
36         for (auto &qq : ql[i]) {
37             for (int j = qq.l; j <= qq.r; ++j) {
38                 if (qq.id > 0) res[qq.id] += suml.query(a[j]+1);
39                 else res[-qq.id] -= suml.query(a[j]+1);
40             }
41         }
42     }
43     sumr.init(n);
44     for (int i = n; i; --i) {
45         sumr.add(a[i], 1);
46         for (auto &qq : qr[i]) {
47             for (int j = qq.l; j <= qq.r; ++j) {
48                 if (qq.id > 0) res[qq.id] += sumr.query(a[j]-1);

```

```

49         else res[-qq.id] -= sumr.query(a[j]-1);
50     }
51 }
52 }
53 for (int i = 1; i <= m; ++i) {
54     res[i] += res[i-1];
55     ans[q[i].id] = res[i];
56 }
57 for (int i = 1; i <= m; ++i) write(ans[i]), putchar('\n');
58 }

```

## 24 ST 表

### 24.1 一维

```

1  template <typename T, typename U = std::greater<T>>
2  struct ST {
3      static const int NN = (int)log2(N)+3;
4      static const T INF = 1e9;
5      int lg2[N];
6      U cmp = U();
7      T rmq[N][NN];
8      ST() {
9          fill(rmq[0], rmq[0]+N*NN, cmp(-INF, +INF) ? INF : -INF);
10         for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1;
11     }
12     T& operator [] (const int &i) { return rmq[i][0]; }
13     void init(const T &val = 0) { fill(rmq[0], rmq[0]+N*NN, val); }
14     T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
15     // rmq[i][j] ==> [i, i+2^j-1]
16     void build(T a[], const int &n) {
17         for (int i = n; i; --i) {
18             rmq[i][0] = a[i];
19             for (int j = 1; j <= lg2[n-i+1]; ++j)
20                 rmq[i][j] = mv(rmq[i][j-1], rmq[i+(1<<(j-1))][j-1]);
21         }
22     }
23     T query(const int &l, const int &r) {
24         if (l > r) return query(r, l);
25         int k = lg2[r-l+1];
26         return mv(rmq[l][k], rmq[r-(1<<k)+1][k]);
27     }
28 };

```

### 24.2 二维

$O(nm \log n \log m)$

```

1  template <typename T, typename U = std::greater<T>>
2  struct ST {
3      static const int NN = (int)log2(N)+3;
4      static const T INF = 1e9;
5      U cmp = U();
6      T rmq[N][N][NN][NN]; // rmq[i][j][k][l] [i, j] [i+2^k-1, j+2^l-1]

```



```

7  ST() { init(); }
8  ST(const T &val) { init(val); }
9  T& operator [] (const int &i) { return rmq[i][0]; }
10 void init(){ fill(rmq[0][0][0], rmq[0][0][0]+N*N*NN*NN, cmp(-INF, +
    INF) ? INF : -INF); }
11 void init(const T &val) { fill(rmq[0][0][0], rmq[0][0][0]+N*N*NN*NN
    , val); }
12 T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
13 void build(T a[N][N], const int &n, const int &m) {
14     for (int k = 0; k <= log_2[n]; ++k)
15         for (int l = 0; l <= log_2[m]; ++l)
16             for (int i = 1; i+(1<<k)-1 <= n; ++i)
17                 for (int j = 1; j+(1<<l)-1 <= m; ++j) {
18                     T &cur = rmq[i][j][k][l];
19                     if (!k && !l) cur = a[i][j];
20                     else if (!l) cur = mv(rmq[i][j][k-1][l], rmq[i+(1<<(k-1))][j][k
                        -1][l]);
21                     else cur = mv(rmq[i][j][k][l-1], rmq[i][j+(1<<(l-1))][k][l-1]);
22                 }
23 }
24 T query(const int &r1, const int &c1, const int &r2, const int &c2)
25 {
26     int k = log_2[r2-r1+1], l = log_2[c2-c1+1];
27     return mv(mv(rmq[r1][c1][k][l], rmq[r2-(1<<k)+1][c2-(1<<l)+1][k][
        l]),
28         mv(rmq[r2-(1<<k)+1][c1][k][l], rmq[r1][c2-(1<<l)+1][k][l]));
29 };

```

## 24.3 反向 ST

```

1  template <typename T, typename U = std::greater<T>>
2  struct rST {
3      static const int NN = (int)log2(N)+3;
4      static const T INF = 1e9;
5      int n;
6      int lg2[N];
7      U cmp = U();
8      T rmq[N][NN]; // rmq[i][j] ==> [i, i+2^j-1]
9      rST() { for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1; }
10     T& operator [] (const int &i) { return rmq[i][0]; }
11     T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
12     void init(const int &n, const T &val = 0) {
13         n = n;
14         for (int i = 1; i <= n; ++i) fill(rmq[i], rmq[i]+NN, val);
15     }
16     void update(const int &l, const int &r, const T &k) {
17         if (l > r) return void(update(r, l, k));
18         int b = lg2[r-l+1];
19         rmq[l][b] = mv(rmq[l][b], k);
20         rmq[r-(1<<b)+1][b] = mv(rmq[r-(1<<b)+1][b], k);
21     }
22     void build() {
23         for (int i = lg2[n]; i >= 0; --i) {

```

```

24     for (int l = 1, r; l <= n; ++l) {
25         r = l+(1<<i);
26         if (r <= n) rmq[r][i] = mv(rmq[r][i], rmq[l][i+1]);
27         rmq[l][i] = mv(rmq[l][i], rmq[l][i+1]);
28     }
29 }
30 }
31 T query(const int &l, const int &r) {
32     if (l > r) return query(r, l);
33     int b = lg2[r-l+1];
34     return mv(rmq[l][b], rmq[r-(1<<b)+1][b]);
35 }
36 };

```

## 25 并查集

```

1 struct DSU {
2     int fa[N];
3     void init(int sz) { for (int i = 0; i <= sz; ++i) fa[i] = i; }
4     int get(int s) { return s == fa[s] ? s : fa[s] = get(fa[s]); }
5     int& operator [] (int i) { return fa[get(i)]; }
6     bool merge(int x, int y) { // merge x to y
7         int fx = get(x), fy = get(y);
8         if (fx == fy) return false;
9         fa[fx] = fy; return true;
10    }
11 } dsu;

```

加上数量

```

1 struct DSU {
2     int fa[N], num[N];
3     void init(int sz) { for (int i = 0; i <= sz; ++i) fa[i] = i, num[i]
4         = 1; }
5     int get(int s) { return s == fa[s] ? s : fa[s] = get(fa[s]); }
6     int& operator [] (int i) { return fa[get(i)]; }
7     bool merge(int x, int y) {
8         int fx = get(x), fy = get(y);
9         if (fx == fy) return false;
10        if (num[fx] >= num[fy]) num[fx] += num[fy], fa[fy] = fx;
11        else num[fy] += num[fx], fa[fx] = fy;
12        return true;
13    }
14 } dsu;

```

## Part IV 字符串

### 26 回文字符串 | manacher 算法

从 0 开始, 第  $i$  位对应  $p[i*2+2]$

```

1 inline int manacher(const char *str, char *buf, int *p) {
2     int str_len = strlen(str), buf_len = 2;
3     buf[0] = buf[1] = '#';
4     for(int i = 0; i < str_len; ++i)
5         buf[buf_len++] = str[i], buf[buf_len++] = '#';
6
7     int mx = 0, id, ans = 0;
8     for(int i = 1; i < buf_len; ++i) {
9         if(i <= mx) p[i] = min(p[id*2-i], mx-i);
10        else p[i] = 1;
11        while(buf[i-p[i]] == buf[i+p[i]]) p[i]++;
12        if(i+p[i] > mx) mx = i+p[i], id = i;
13        ans = max(ans, p[i]-1);
14    }
15    return ans;
16 }

```

## 26.1 判断 $s[l, r]$ 是否为回文

```

1 p[l+r+2]-1 >= r-l+1

```

## 27 KMP

```

1 inline void get_next(const string &s, int nex[]) { get_next(s.c_str(),
2     , nex); }
3 inline void get_next(const char *s, int nex[]) {
4     nex[0] = nex[1] = 0;
5     for (int i = 1, j = 0, l = strlen(s); i < l; ++i) {
6         while (j && s[i] != s[j]) j = nex[j];
7         nex[i+1] = s[i] == s[j] ? ++j : 0;
8     }
9 }
10 inline void kmp(const string &s1, const string &s2, int nex[]) { kmp(
11     s1.c_str(), s2.c_str(), nex); }
12 inline void kmp(const char *s1, const char *s2, int nex[]) {
13     for (int i = 0, j = 0, l1 = strlen(s1), l2 = strlen(s2); i < l1; ++
14         i){
15         while (j && s1[i] != s2[j]) j = nex[j];
16         if (s1[i] == s2[j]) ++j;
17         if (j == l2) {
18             cout << i-l2+2 << endl;
19             j = nex[j];
20         }
21     }
22 }

```

```

1 inline void get_next(const string &s, int nex[]) {
2     nex[0] = nex[1] = 0;
3     for (int i = 1, j = 0; i < (int)s.size(); ++i) {
4         while (j && s[i] != s[j]) j = nex[j];
5         nex[i+1] = s[i] == s[j] ? ++j : 0;

```

```

6   }
7   }
8
9   inline void kmp(const string &s1, const string &s2, int nex[]) {
10      for (int i = 0, j = 0; i < (int)s1.size(); ++i) {
11          while (j && s1[i] != s2[j]) j = nex[j];
12          if (s1[i] == s2[j]) ++j;
13          if (j == (int)s2.size()) {
14              cout << i-s2.size()+2 << endl;
15              j = nex[j];
16          }
17      }
18  }

```

## 28 扩展 KMP|Z 函数

```

1   inline void GetNext(char *s, int *_nex) {
2       int len = strlen(s);
3       int a = 0, p = 0;
4       _nex[0] = len;
5       for (int i = 1; i < len; ++i) {
6           if (i >= p || i+_nex[i-a] >= p) {
7               if (i > p) p = i;
8               while (p < len && s[p] == s[p-i]) ++p;
9               a = i;
10              _nex[i] = p-i;
11          } else {
12              _nex[i] = _nex[i-a];
13          }
14      }
15  }
16
17  inline void GetExtend(char *s, char *ss, int *_ext, int *_nex) {
18      int lens = strlen(s), lenss = strlen(ss);
19      int a = 0, p = 0;
20      for (int i = 0; i < lens; ++i) {
21          if (i >= p || i+_nex[i-a] >= p) {
22              if (i > p) p = i;
23              while (p < lens && p-i < lenss && s[p] == ss[p-i]) ++p;
24              a = i;
25              _ext[i] = p-i;
26          } else {
27              _ext[i] = _nex[i-a];
28          }
29      }
30  }

```

## 29 后缀数组 |SA

### 29.1 $O(n\log^2 n)$

```

1 int sa[N], rk[N<<1], height[N];
2 template <typename T> // s start from 1
3 inline void SA(const T *s, const int &n) {
4     static int oldrk[N<<1];
5     memset(rk+n+1, 0, sizeof(int)*n);
6     for (int i = 1; i <= n; ++i) rk[i] = s[i];
7     for (int w = 1; w <= n; w <= 1) {
8         iota(sa+1, sa+n+1, 1);
9         sort(sa+1, sa+n+1, &{
10             return rk[x] == rk[y] ? rk[x+w] < rk[y+w] : rk[x] < rk[y];
11         });
12         memcpy(oldrk+1, rk+1, sizeof(int)*2*n);
13         for (int p = 0, i = 1; i <= n; ++i) {
14             if (oldrk[sa[i]] == oldrk[sa[i-1]] &&
15                 oldrk[sa[i]+w] == oldrk[sa[i-1]+w]) {
16                 rk[sa[i]] = p;
17             } else {
18                 rk[sa[i]] = ++p;
19             }
20         }
21     }
22     for (int i = 1, k = 0; i <= n; ++i) {
23         if (k) --k;
24         while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;
25         height[rk[i]] = k;
26     }
27 }

```

## 29.2 $O(n)$

```

1 namespace SuffixArray {
2
3 int sa[N], rk[N], ht[N];
4 bool t[N << 1];
5
6 inline bool islms(const int i, const bool *t) { return i > 0 && t[i]
    && !t[i - 1]; }
7
8 template <class T>
9 inline void sort(T s, int *sa, const int len, const int sz, const int
    sigma, bool *t, int *b, int *cb, int *p) {
10     memset(b, 0, sizeof(int) * sigma);
11     memset(sa, -1, sizeof(int) * len);
12     for (register int i = 0; i < len; i++) b[static_cast<int>(s[i])]++;
13     cb[0] = b[0];
14     for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i];
15     for (register int i = sz - 1; i >= 0; i--) sa[--cb[static_cast<int>
        >(s[p[i]])]] = p[i];
16     for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i -
        1];
17     for (register int i = 0; i < len; i++)
18         if (sa[i] > 0 && !t[sa[i] - 1])
19             sa[cb[static_cast<int>(s[sa[i] - 1])]++] = sa[i] - 1;
20     cb[0] = b[0];
21     for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i];

```

```

22   for (register int i = len - 1; i >= 0; i--)
23       if (sa[i] > 0 && t[sa[i] - 1])
24           sa[--cb[static_cast<int>(s[sa[i] - 1])]] = sa[i] - 1;
25 }
26
27 template <class T>
28 inline void sais(T s, int *sa, const int len, bool *t, int *b, int *
    b1, const int sigma) {
29     register int i, j, x, p = -1, cnt = 0, sz = 0, *cb = b + sigma;
30     for (t[len - 1] = 1, i = len - 2; i >= 0; i--) t[i] = s[i] < s[i +
        1] || (s[i] == s[i + 1] && t[i + 1]);
31     for (i = 1; i < len; i++)
32         if (t[i] && !t[i - 1])
33             b1[sz++] = i;
34     sort(s, sa, len, sz, sigma, t, b, cb, b1);
35     for (i = sz = 0; i < len; i++)
36         if (islms(sa[i], t))
37             sa[sz++] = sa[i];
38     for (i = sz; i < len; i++) sa[i] = -1;
39     for (i = 0; i < sz; i++) {
40         for (x = sa[i], j = 0; j < len; j++) {
41             if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) {
42                 cnt++, p = x;
43                 break;
44             } else if (j > 0 && (islms(x + j, t) || islms(p + j, t))) {
45                 break;
46             }
47         }
48         sa[sz + (x >>= 1)] = cnt - 1;
49     }
50     for (i = j = len - 1; i >= sz; i--)
51         if (sa[i] >= 0)
52             sa[j--] = sa[i];
53     register int *s1 = sa + len - sz, *b2 = b1 + sz;
54     if (cnt < sz)
55         sais(s1, sa, sz, t + len, b, b1 + sz, cnt);
56     else
57         for (i = 0; i < sz; i++) sa[s1[i]] = i;
58     for (i = 0; i < sz; i++) b2[i] = b1[sa[i]];
59     sort(s, sa, len, sz, sigma, t, b, cb, b2);
60 }
61
62 template <class T>
63 inline void getHeight(T s, int n) {
64     for (register int i = 1; i <= n; i++) rk[sa[i]] = i;
65     register int j = 0, k = 0;
66     for (register int i = 0; i < n; ht[rk[i++]] = k)
67         for (k ? k-- : 0, j = sa[rk[i] - 1]; s[i + k] == s[j + k]; k++)
68             ;
69 }
70
71 template <class T> // s start from 0
72 inline void init(T s, const int len, const int sigma = 128) {
73     sais(s, sa, len + 1, t, rk, ht, sigma);
74     getHeight(s, len);
75     for (int i = 1; i <= len; ++i) ++sa[i];
76     for (int i = len; i; --i) rk[i] = rk[i-1];

```

```

77 }
78
79 } // namespace SuffixArray

```

## 30 字典树

```

1 struct TireTree {
2     static const int NN = 5e5+7;
3     static const int SZ = 26;
4     char beg;
5     int nex[NN][SZ], num[NN], cnt;
6     bool exist[NN];
7     TireTree(char _beg = 'a') : beg(_beg) { clear(); }
8     void clear() {
9         memset(nex, 0, sizeof nex);
10        memset(num, 0, sizeof num);
11        memset(exist, 0, sizeof exist);
12        cnt = 0;
13    }
14    void insert(const char *s) {
15        int len = strlen(s), p = 0;
16        for (int i = 0, c; i < len; ++i) {
17            c = s[i]-beg;
18            if (!nex[p][c]) nex[p][c] = ++cnt;
19            p = nex[p][c];
20            ++num[p];
21        }
22        exist[p] = true;
23    }
24    bool find(const char *s) {
25        int len = strlen(s), p = 0;
26        for (int i = 0, c; i < len; ++i) {
27            c = s[i]-beg;
28            if (!nex[p][c]) return false;
29            p = nex[p][c];
30        }
31        return exist[p];
32    }
33    int count(const char *s) {
34        int len = strlen(s), p = 0;
35        for (int i = 0, c; i < len; ++i) {
36            c = s[i]-beg;
37            if (!nex[p][c]) return 0;
38            p = nex[p][c];
39        }
40        return num[p];
41    }
42    void insert(const string &s) { insert(s.c_str()); }
43    bool find(const string &s) { return find(s.c_str()); }
44    int count(const string &s) { return count(s.c_str()); }
45 };

```

## 31 AC 自动机

如需构造可重建 AC 自动机，每次构造建一个 nex 数组的拷贝

```

1 struct Aho_Corasick_Automaton {
2     static const int NN = 5e6+7;
3     static const int SZ = 26;
4     char beg;
5     int nex[NN][SZ], num[NN], fail[NN], cnt;
6     Aho_Corasick_Automaton(const char &_beg = 'a') : beg(_beg) {}
7     void clear() {
8         memset(nex, 0, sizeof(nex[0])*(cnt+1));
9         memset(num, 0, sizeof(int)*(cnt+1));
10        memset(fail, 0, sizeof(int)*(cnt+1));
11        cnt = 0;
12    }
13    void insert(const char *s) {
14        int len = strlen(s), p = 0;
15        for (int i = 0, c; i < len; ++i) {
16            c = s[i]-beg;
17            if (!nex[p][c]) nex[p][c] = ++cnt;
18            p = nex[p][c];
19        }
20        ++num[p];
21    }
22    void build() {
23        static queue<int> q;
24        for (int i = 0; i < SZ; ++i) if (nex[0][i]) q.push(nex[0][i]);
25        while (q.size()) {
26            int u = q.front();
27            q.pop();
28            for (int i = 0; i < SZ; ++i) {
29                if (nex[u][i]) {
30                    fail[nex[u][i]] = nex[fail[u]][i];
31                    q.push(nex[u][i]);
32                } else {
33                    nex[u][i] = nex[fail[u]][i];
34                }
35            }
36        }
37    }
38    int query(const char *s) {
39        int len = strlen(s), p = 0, res = 0;
40        for (int i = 0; i < len; ++i) {
41            p = nex[p][s[i]-beg];
42            for (int t = p; t && ~num[t]; t = fail[t]) {
43                res += num[t];
44                num[t] = -1;
45            }
46        }
47        return res;
48    }
49 };

```

```

1 struct Aho_Corasick_Automaton {
2     static const int NN = 2e5+7;
3     static const int SZ = 26;

```



```

4 char beg;
5 int cnt;
6 int nex[NN][SZ], fail[NN], vis[NN];
7 Aho_Corasick_Automaton(const char &_beg = 'a') : beg(_beg) {}
8 void clear() {
9     memset(nex, 0, sizeof(nex[0])*(cnt+1));
10    memset(fail, 0, sizeof(int)*(cnt+1));
11    memset(vis, 0, sizeof(int)*(cnt+1));
12    cnt = 0;
13 }
14 int insert(const char *s) {
15     int len = strlen(s), p = 0;
16     for (int i = 0, c; i < len; ++i) {
17         c = s[i]-beg;
18         if (!nex[p][c]) nex[p][c] = ++cnt;
19         p = nex[p][c];
20     }
21     return p;
22 }
23 void build() {
24     static queue<int> q;
25     for (int i = 0; i < SZ; ++i) if (nex[0][i]) q.push(nex[0][i]);
26     while (q.size()) {
27         int u = q.front();
28         q.pop();
29         for (int i = 0; i < SZ; ++i) {
30             if (nex[u][i]) {
31                 fail[nex[u][i]] = nex[fail[u]][i];
32                 q.push(nex[u][i]);
33             } else {
34                 nex[u][i] = nex[fail[u]][i];
35             }
36         }
37     }
38 }
39 void query(char *s) {
40     static int deg[NN];
41     static queue<int> q;
42
43     int len = strlen(s);
44     for (int i = 0, p = 0; i < len; ++i) {
45         p = nex[p][s[i]-beg];
46         ++vis[p];
47         // for (int t = p; t; t = fail[t]) ++vis[t];
48     }
49     for (int i = 1; i <= cnt; ++i) ++deg[fail[i]];
50     for (int i = 1; i <= cnt; ++i) if (!deg[i]) q.push(i);
51     while (q.size()) {
52         int u = q.front();
53         q.pop();
54         vis[fail[u]] += vis[u];
55         if (--deg[fail[u]] == 0) q.push(fail[u]);
56     }
57 }
58 } ac;

```

## Part V

## 图论 | 树论

## 32 树的重心

```

1 void treedp(int cur, int fa) {
2     s[cur] = c[cur];
3     for(int i = fir[cur]; i; i = nex[i]) {
4         if(e[i] == fa) continue;
5         treedp(e[i], cur);
6         s[cur] += s[e[i]];
7         maxs[cur] = max(maxs[cur], s[e[i]]);
8     }
9     maxs[cur] = max(maxs[cur], sum-s[cur]);
10 }

```

## 33 最大团

最大独立集数 = 补图的最大团

```

1 struct MaxClique {
2     vector<int> res, tmp, cnt;
3     bool dfs(int p) {
4         for (int i = p+1, flag; i <= n; ++i) {
5             if (cnt[i]+tmp.size() <= res.size()) return false;
6             if (!g[p][i]) continue;
7             flag = 1;
8             for (int j : tmp)
9                 if (!g[i][j]) flag = 0;
10            if (!flag) continue;
11            tmp.push_back(i);
12            if (dfs(i)) return true;
13            tmp.pop_back();
14        }
15        if (tmp.size() > res.size()) {
16            res = tmp;
17            return true;
18        }
19        return false;
20    }
21    void solve() {
22        vector<int>(n+1, 0).swap(cnt);
23        vector<int>().swap(res);
24        for (int i = n; i; --i) {
25            vector<int>(1, i).swap(tmp);
26            dfs(i);
27            cnt[i] = res.size();
28        }
29    }
30 } MC;

```

## 34 稳定婚姻匹配

```

1  template <typename T = int> struct Stable_Marriage {
2      int t[N], b[N], g[N], rkb[N][N], rkg[N][N];
3      T wb[N][N], wg[N][N];
4      queue<int> q;
5      void init(const int &n) {
6          queue<int>().swap(q);
7          memset(t, 0, sizeof(int)*(n+3));
8          memset(b, 0, sizeof(int)*(n+3));
9          memset(g, 0, sizeof(int)*(n+3));
10         for (int i = 1; i <= n; ++i) {
11             q.push(i);
12             for (int j = 1; j <= n; ++j)
13                 rkb[i][j] = rkg[i][j] = j;
14             sort(rkb[i]+1, rkb[i]+n+1,
15                 &);
16             //sort(rkg[i]+1, rkg[i]+n+1,
17                 //    &);
18         }
19     }
20     bool match(const int &x, const int &y) {
21         if (g[y]) {
22             if (wg[y][x] < wg[y][g[y]]) return false;
23             b[g[y]] = 0;
24             q.push(g[y]);
25         }
26         b[x] = y; g[y] = x;
27         return true;
28     }
29     void gale_shapely(const int &n) {
30         init(n);
31         while (q.size()) {
32             int x = q.front(); q.pop();
33             int y = rkb[x][++t[x]];
34             if (!match(x, y)) q.push(x);
35         }
36     }
37 };

```

## 35 最小生成树

Prim

```

1  inline void prim() {
2      fill(dis, dis+n+1, INF);
3      dis[1] = 0;
4      for(int t = 1; t <= n; ++t)
5          {
6              int mini = 0;
7              for(int i = 1; i <= n; ++i)
8                  if(!vis[i] && dis[i] < dis[mini])
9                      mini = i;
10             vis[mini] = 1;
11             ans += dis[mini];

```

```

12     for(int i = 1; i <= n; ++i)
13         if(!vis[i]) dis[i] = min(dis[i], calc(mini, i));
14     }
15 }

```

Kruskal (略)

## 36 二分图

### 36.1 二分图匹配

匈牙利算法

```

1 bool check(int u) {
2     for (int v : e[u]) {
3         if (vis[v]) continue;
4         vis[v] = 1;
5         if (!co[v] || check(co[v])) {
6             co[v] = u;
7             return true;
8         }
9     }
10    return false;
11 }
12
13 inline int solve() {
14     int res = 0;
15     memset(co, 0, sizeof co);
16     for (int i = 1; i <= n; ++i) {
17         memset(vis, 0, sizeof(int)*(n+3));
18         res += check(i);
19     }
20     return res;
21 }

```

### 36.2 二分图最小顶点覆盖

定义：假如选了一个点就相当于覆盖了以它为端点的所有边。最小顶点覆盖就是选择最少的点来覆盖所有的边。

定理：最小顶点覆盖等于二分图的最大匹配。

### 36.3 最大独立集

定义：选出一些顶点使得这些顶点两两不相邻，则这些点构成的集合称为独立集。找出一个包含顶点数最多的独立集称为最大独立集。

定理：最大独立集 = 所有顶点数 - 最小顶点覆盖 = 所有顶点数 - 最大匹配

## 37 最近公共祖先 |LCA

### 37.1 倍增

```

1 struct LCA {
2     static const int NN = (int)log2(N)+3;
3     int f[N][NN], d[N], lg2[N];
4     LCA() { for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1; }
5     template <typename TT>
6     void build(const TT e[], const int &u = 1, const int &fa = 0) {
7         d[u] = d[fa]+1;
8         f[u][0] = fa;
9         for (int i = 1; (1<<i) <= d[u]; ++i)
10             f[u][i] = f[f[u][i-1]][i-1];
11         for (auto v : e[u]) if (v != fa)
12             build(e, v, u);
13     }
14     int get(int x, int y) {
15         if (d[x] < d[y]) swap(x, y);
16         while (d[x] > d[y])
17             x = f[x][lg2[d[x]-d[y]]];
18         if (x == y) return x;
19         for (int i = lg2[d[x]]; i >= 0; --i)
20             if (f[x][i] != f[y][i])
21                 x = f[x][i], y = f[y][i];
22         return f[x][0];
23     }
24 };

```

## 37.2 帶权 LCA

```

1 template <typename T>
2 struct LCA {
3     static const int NN = (int)log2(N)+3;
4     int f[N][NN], d[N], lg2[N];
5     T w[N][NN], init_val = 0;
6     LCA() {
7         for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1;
8         init();
9     }
10    // set sum or min or max, and don't forget to set init_val
11    T update(const T &x, const T &y) { return x+y; }
12    void init(const int &n = N-1) {
13        fill(w[0], w[0]+(n+1)*NN, init_val);
14    }
15    template <typename TT>
16    void build(const TT e[], const int &u = 1, const int &fa = 0) {
17        d[u] = d[fa]+1;
18        f[u][0] = fa;
19        for (int i = 1; (1<<i) <= d[u]; ++i) {
20            f[u][i] = f[f[u][i-1]][i-1];
21            w[u][i] = update(w[u][i-1], w[f[u][i-1]][i-1]);
22        }
23        for (auto v : e[u]) if (v.first != fa) {
24            w[v.first][0] = v.second;
25            build(e, v.first, u);
26        }
27    }
28    T get(int x, int y) {

```

```

29     T res = init_val;
30     if (d[x] < d[y]) swap(x, y);
31     while (d[x] > d[y]) {
32         res = update(res, w[x][lg2[d[x]-d[y]]]);
33         x = f[x][lg2[d[x]-d[y]]];
34     }
35     if (x == y) return res;
36     for (int i = lg2[d[x]]; i >= 0; --i)
37         if (f[x][i] != f[y][i]) {
38             res = update(res, w[x][i]);
39             res = update(res, w[y][i]);
40             x = f[x][i], y = f[y][i];
41         }
42     return update(res, update(w[x][0], w[y][0]));
43 }
44 };

```

## 38 树上差分

```

1  template <typename T>
2  struct Tree {
3      T val[N];
4      void update_point(const int &x, const int &y, const T &k) {
5          int _lca = lca(x, y);
6          val[x] += k; val[y] += k;
7          val[_lca] -= k; val[f[_lca][0]] -= k;
8      }
9      void update_edge(const int &x, const int &y, const T &k) {
10         int _lca = lca(x, y);
11         val[x] += k; val[y] += k; val[_lca] -= 2*k;
12     }
13     void dfs(const int &u = 1, const int &fa = 0) {
14         for (int v : e[u]) if (v != fa) {
15             dfs(v, u);
16             val[u] += val[v];
17         }
18     }
19 };

```

## 39 树链剖分

```

1  template <typename T>
2  struct HLD {
3      int dfn;
4      int fa[N], d[N], num[N], son[N], id[N], tp[N];
5      T init_val[N];
6      SegmentTree<T> ST;
7      template <typename Edge, typename TT>
8      void build(const Edge e[], const TT a[], const int &n, const int &
9          rt = 1) {
10         fa[rt] = dfn = 0;
11         dfs1(e, rt);

```

```

11     dfs2(e, rt);
12     for (int i = 1; i <= n; ++i)
13         init_val[id[i]] = a[i];
14     ST.build(init_val, n);
15 }
16 template <typename Edge>
17 void dfs1(const Edge e[], const int &u = 1) {
18     d[u] = d[fa[u]]+1;
19     num[u] = 1;
20     son[u] = 0;
21     for (const int &v : e[u]) if (v != fa[u]) {
22         fa[v] = u;
23         dfs1(e, v);
24         num[u] += num[v];
25         if (num[v] > num[son[u]]) son[u] = v;
26     }
27 }
28 template <typename Edge>
29 void dfs2(const Edge e[], const int &u = 1) {
30     tp[u] = son[fa[u]] == u ? tp[fa[u]] : u;
31     id[u] = ++dfn;
32     if (son[u]) dfs2(e, son[u]);
33     for (const int &v : e[u]) if (v != son[u] && v != fa[u])
34         dfs2(e, v);
35 }
36 void add_sons(const int &x, const T &k) { ST.add(id[x], id[x]+num[x]-1, k); }
37 void add(int x, int y, const T &k, const int &is_edge = 0) {
38     while (tp[x] != tp[y]) {
39         if (d[tp[x]] < d[tp[y]]) swap(x, y);
40         ST.add(id[tp[x]], id[x], k);
41         x = fa[tp[x]];
42     }
43     if (d[x] > d[y]) swap(x, y);
44     ST.add(id[x], id[y], k);
45     if (is_edge) ST.add(id[x], -k);
46 }
47 T query_sons(const int &x) { return ST.query(id[x], id[x]+num[x]-1); }
48 T query(const int &x) { return ST.query(id[x]); }
49 T query(int x, int y) {
50     T res = 0;
51     while (tp[x] != tp[y]) {
52         if (d[tp[x]] < d[tp[y]]) swap(x, y);
53         res += ST.query(id[tp[x]], id[x]);
54         x = fa[tp[x]];
55     }
56     if (d[x] > d[y]) swap(x, y);
57     return res+ST.query(id[x], id[y]);
58 }
59 };

```

## 40 网络流

### 40.1 最大流

#### 40.1.1 EK

$O(nm^2)$

```

1  template <typename T>
2  struct EK {
3      struct Edge {
4          int v, nex;
5          T w;
6      } e[M<<1];
7      int tot = 0, n;
8      int fir[N], vis[N], pre[N];
9      T incf[N];
10     T work(const int &s, const int &t) {
11         T res = 0;
12         while (bfs(s, t)) {
13             int u = t, id;
14             while (u != s) {
15                 id = pre[u];
16                 e[id].w -= incf[t];
17                 e[id^1].w += incf[t];
18                 u = e[id^1].v;
19             }
20             res += incf[t];
21         }
22         return res;
23     }
24     void init(const int &sz) {
25         n = sz;
26         tot = 0;
27         memset(fir, -1, sizeof(int)*(n+3));
28     }
29     void add_edge(const int &u, const int &v, const T &w) {
30         e[tot] = {v, fir[u], w}; fir[u] = tot++;
31         e[tot] = {u, fir[v], 0}; fir[v] = tot++;
32     }
33     bool bfs(const int &s, const int &t) {
34         queue<int> q;
35         memset(vis, 0, sizeof(int)*(n+3));
36         q.push(s);
37         vis[s] = 1;
38         incf[s] = INF;
39         while (q.size()) {
40             int u = q.front();
41             q.pop();
42             for (int i = fir[u], v; i != -1; i = e[i].nex) {
43                 v = e[i].v;
44                 if (vis[v] || !e[i].w) continue;
45                 incf[v] = min(incf[u], e[i].w);
46                 pre[v] = i;
47                 if (v == t) return true;
48                 q.push(v);
49                 vis[v] = 1;
50             }

```



```

51     }
52     return false;
53 }
54 };

```

### 40.1.2 Dinic

普通情况下  $O(n^2m)$  二分图中  $O(\sqrt{nm})$

```

1  template <typename T>
2  struct Dinic {
3      struct EDGE {
4          int v, nex;
5          T w;
6          EDGE(const int &_v, const int &_nex, const T &_w) : v(_v), nex(
              _nex), w(_w) {}
7      };
8      vector<EDGE> e;
9      int n, s, t;
10     int fir[N], dep[N], cur[N];
11     Dinic() { e.reserve(N<<2); }
12     T work(const int &_s, const int &_t) {
13         s = _s; t = _t;
14         T maxflow = 0, flow;
15         while (bfs())
16             while ((flow = dfs(s, INF)))
17                 maxflow += flow;
18         return maxflow;
19     }
20     void init(const int &n) {
21         n = n;
22         e.clear();
23         memset(fir, -1, sizeof(int)*(n+3));
24     }
25     void add_edge(const int &u, const int &v, const T &w) {
26         e.emplace_back(v, fir[u], w); fir[u] = e.size()-1;
27         e.emplace_back(u, fir[v], 0); fir[v] = e.size()-1;
28     }
29     bool bfs() {
30         queue<int> q;
31         memset(dep, 0, sizeof(int)*(n+3));
32         q.push(s);
33         dep[s] = 1;
34         for (int i = 0; i <= n; ++i) cur[i] = fir[i];
35         while (q.size()) {
36             int u = q.front();
37             q.pop();
38             for (int i = fir[u], v; i != -1; i = e[i].nex) {
39                 v = e[i].v;
40                 if (dep[v] || !e[i].w) continue;
41                 dep[v] = dep[u]+1;
42                 if (v == t) return true;
43                 q.push(v);
44             }
45         }
46         return false;
47     }

```

```

48 T dfs(const int &u, const T &flow) {
49     if (!flow || u == t) return flow;
50     T rest = flow, now;
51     for (int &i = cur[u], v; i != -1; i = e[i].nex) {
52         v = e[i].v;
53         if (dep[v] != dep[u]+1 || !e[i].w) continue;
54         now = dfs(v, min(rest, e[i].w));
55         if (!now) {
56             dep[v] = 0;
57         } else {
58             e[i].w -= now;
59             e[i^1].w += now;
60             rest -= now;
61             if (rest == flow) break;
62         }
63     }
64     return flow - rest;
65 }
66 };

```

## 40.2 最小割

最小割等价最大流

## 40.3 费用流

### 40.3.1 MCMF

```

1  template <typename T>
2  struct MCMF {
3      struct Edge {
4          int v, nex;
5          T w, c; // edge wight and cost
6          Edge(const int &_v, const int &_nex, const T &_w, const T &_c) \
7              : v(_v), nex(_nex), w(_w), c(_c) {}
8      };
9      vector<Edge> e;
10     int n, s, t;
11     int fir[N], vis[N], pre[N];
12     T incf[N], dis[N];
13     void init(const int &n) {
14         n = _n;
15         e.clear();
16         e.reserve(N<<4);
17         memset(fir, -1, sizeof(int)*(n+3));
18     }
19     void add_edge(const int &u, const int &v, const T &w, const T &c) {
20         e.emplace_back(v, fir[u], w, c); fir[u] = e.size()-1;
21         e.emplace_back(u, fir[v], 0, -c); fir[v] = e.size()-1;
22     }
23     pair<T, T> work(const int &s, const int &t) {
24         s = _s; t = _t;
25         T maxflow = 0, mincost = 0;
26         while (spfa()) {
27             for (int u = t, id; u != s; u = e[id^1].v) {

```

```

28         id = pre[u];
29         e[id].w -= incf[t];
30         e[id^1].w += incf[t];
31         mincost += incf[t]*e[id].c;
32     }
33     maxflow += incf[t];
34 }
35 return {maxflow, mincost};
36 }
37 bool spfa() {
38     queue<int> q;
39     memset(dis, 0x3f, sizeof(T)*(n+3));
40     memset(vis, 0, sizeof(int)*(n+3));
41     q.push(s);
42     dis[s] = 0;
43     incf[s] = INF;
44     incf[t] = 0;
45     while (q.size()) {
46         int u = q.front();
47         q.pop();
48         vis[u] = 0;
49         for (int i = fir[u], v; i != -1; i = e[i].nex) {
50             v = e[i].v;
51             if (!e[i].w || dis[v] <= dis[u]+e[i].c) continue;
52             dis[v] = dis[u]+e[i].c;
53             incf[v] = min(incf[u], e[i].w);
54             pre[v] = i;
55             if (vis[v]) continue;
56             q.push(v);
57             vis[v] = 1;
58         }
59     }
60     return incf[t];
61 }
62 };

```

### 40.3.2 ZKW\_SPFA

```

1  template <typename T>
2  struct ZKW_SPFA {
3      struct Edge {
4          int v, nex;
5          T w, c; // edge wight and cost
6          Edge(const int &_v, const int &_nex, const T &_w, const T &_c) \
7              : v(_v), nex(_nex), w(_w), c(_c) {}
8      };
9      vector<Edge> e;
10     int n, s, t;
11     int fir[N], vis[N];
12     T maxflow, mincost;
13     T dis[N];
14     ZKW_SPFA() { e.reserve(N<<4); }
15     void init(const int &n) {
16         n = n;
17         maxflow = mincost = 0;
18         e.clear();

```

```

19     memset(fir, -1, sizeof(int)*(n+3));
20 }
21 void add_edge(const int &u, const int &v, const T &w = 1, const T &
    c = 0) {
22     e.emplace_back(v, fir[u], w, c); fir[u] = e.size()-1;
23     e.emplace_back(u, fir[v], 0, -c); fir[v] = e.size()-1;
24 }
25 pair<T, T> work(const int &s, const int &t) {
26     s = _s; t = _t;
27     while (spfa()) {
28         vis[t] = 1;
29         while (vis[t]) {
30             memset(vis, 0, sizeof(int)*(n+3));
31             maxflow += dfs(s, INF);
32         }
33     }
34     return {maxflow, mincost};
35 }
36 private:
37 bool spfa() {
38     memset(dis, 0x3f, sizeof(T)*(n+3));
39     memset(vis, 0, sizeof(int)*(n+3));
40     deque<int> q;
41     q.push_back(t);
42     dis[t] = 0;
43     vis[t] = 1;
44     while (q.size()) {
45         int u = q.front(); q.pop_front();
46         for (int i = fir[u], v; ~i; i = e[i].nex) {
47             v = e[i].v;
48             if (!e[i^1].w || dis[v] <= dis[u]+e[i^1].c) continue;
49             dis[v] = dis[u]+e[i^1].c;
50             if (vis[v]) continue;
51             vis[v] = 1;
52             if (q.size() && dis[v] < dis[q.front()]) q.push_front(v);
53             else q.push_back(v);
54         }
55         vis[u] = 0;
56     }
57     return dis[s] < INF;
58 }
59 T dfs(const int &u, const T &flow) {
60     vis[u] = 1;
61     if (u == t || flow <= 0) return flow;
62     T res, used = 0;
63     for (int i = fir[u], v; ~i; i = e[i].nex) {
64         v = e[i].v;
65         if (vis[v] || !e[i].w || dis[u] != dis[v]+e[i].c) continue;
66         res = dfs(v, min(e[i].w, flow-used));
67         if (!res) continue;
68         mincost += res*e[i].c;
69         e[i].w -= res;
70         e[i^1].w += res;
71         used += res;
72         if (used == flow) break;
73     }
74     return used;

```

```

75 | }
76 | };

```

## 41 最短路

### 41.1 Floyd

### 41.2 Dijkstra

### 41.3 SPFA

```

1 inline void SPFA() {
2     fill(dis+1, dis+n+1, INT_MAX);
3     dis[S] = 0;
4     head = tail = 0;
5     q[++tail] = S;
6     while(head < tail) {
7         int cur = q[++head];
8         for(int i = fir[cur], to, tmp; i; i = nex[i]) {
9             to = ver[i];
10            tmp = dis[cur]+w[i];
11            if(tmp <= dis[to]) continue;
12            dis[to] = tmp;
13            q[++tail] = to;
14        }
15    }
16 }

```

## 42 负环

```

1 // 返回true有负环,返回false没负环
2 inline bool SPFA() {
3     q[++tail] = 1;
4     vis[1] = 1;
5     cnt[1] = 1;
6     dis[1] = 0;
7     while(head < tail) {
8         int cur = q[(++head)%Maxn];
9         vis[cur] = 0;
10        for(int i = fir[cur], to; i; i = nex[i]) {
11            to = ver[i];
12            if(dis[cur]+w[i] < dis[to]) {
13                dis[to] = dis[cur]+w[i];
14                if(!vis[to]) {
15                    q[(++tail)%Maxn] = to;
16                    vis[to] = 1;
17                    if(++cnt[to] > n) return true;
18                }
19            }
20        }
21    }
22    return false;

```

23 }

## 43 割点

```

1 void tarjan(int cur, int fa) {
2     dfn[cur] = low[cur] = ++_dfn;
3     int child = 0;
4     for(auto i : e[cur]) {
5         if(!dfn[i]) {
6             child++;
7             tarjan(i, fa);
8             low[cur] = min(low[cur], low[i]);
9             if(cur != fa && low[i] >= dfn[cur]) flag[cur] = 1;
10        }
11        low[cur] = min(low[cur], dfn[i]);
12    }
13    if(cur == fa && child >= 2) flag[cur] = 1;
14 }

```

## 44 SCC 强连通分量 | Tarjan

```

1 int _dfn, _col, _top;
2 int dfn[N], low[N], vis[N], col[N], sta[N];
3
4 void tarjan(const int &u) {
5     dfn[u] = low[u] = ++_dfn;
6     vis[u] = 1;
7     sta[++_top] = u;
8     for (int v : e[u]) {
9         if (!dfn[v]) {
10            tarjan(v);
11            low[u] = min(low[u], low[v]);
12        } else if (vis[v]) {
13            low[u] = min(low[u], low[v]);
14        }
15    }
16    if (dfn[u] == low[u]) {
17        ++_col;
18        do {
19            col[sta[_top]] = _col;
20            vis[sta[_top]] = 0;
21        } while (sta[_top--] != u);
22    }
23 }

```

## 45 缩点

```

1 void tarjan(int u) {
2     dfn[u] = low[u] = ++_dfn;

```

```

3   vis[u] = 1;
4   sta[++top] = u;
5   for (int v : e[u]) {
6       if (!dfn[v]) {
7           tarjan(v);
8           low[u] = min(low[u], low[v]);
9       } else if (vis[v]) {
10          low[u] = min(low[u], low[v]);
11      }
12  }
13  if (dfn[u] == low[u]) {
14      w_col[++_col] = 0;
15      do {
16          col[sta[top]] = _col;
17          vis[sta[top]] = 0;
18          w_col[_col] += w[sta[top]];
19      } while (sta[top--] != u);
20  }
21 }
22
23 inline void suodian() {
24     for (int i = 1; i <= n; ++i) {
25         if (!dfn[i]) tarjan(i);
26     }
27     for (int i = 1; i <= n; ++i) {
28         for (int j : e[i]) {
29             if (col[i] == col[j]) continue;
30             e_col[col[i]].push_back(col[j]);
31         }
32     }
33 }

```

## 46 2-SAT

### 46.1 SCC Tarjan

$O(n + m)$

```

1  struct TWO_SAT { // node stkr from 0
2      int top, _dfn, _scc;
3      int dfn[N<<1], low[N<<1], stk[N<<1], scc[N<<1], res[N];
4      vector<int> e[N<<1];
5      void init(const int &n) {
6          top = 0;
7          memset(dfn, 0, sizeof(int)*n*2);
8          memset(low, 0, sizeof(int)*n*2);
9          memset(scc, 0, sizeof(int)*n*2);
10         for (int i = 0; i < n<<1; ++i) vector<int>().swap(e[i]);
11     }
12     // if u then v
13     void add_edge(const int &u, const int &v) {
14         e[u].emplace_back(v);
15     }
16     void add_edge(const int &u, const int &uv, const int &v, const int
17         &vv) {
18         e[u<<1^uv].emplace_back(v<<1^vv);
19     }
20 }

```

```

18 }
19 // pt i ==> i<<1 && i<<1/1 ==> 0 && 1
20 inline bool work(const int &n) {
21     for (int i = 0; i <= n<<1; ++i)
22         if (!dfn[i]) tarjan(i);
23     for (int i = 0; i < n; ++i) {
24         if (scc[i<<1] == scc[i<<1|1]) return false;
25         res[i] = scc[i<<1] > scc[i<<1|1];
26     }
27     return true;
28 }
29 void tarjan(const int &u) {
30     dfn[u] = low[u] = ++_dfn;
31     stk[++top] = u;
32     for (int &v : e[u]) {
33         if (!dfn[v]) {
34             tarjan(v);
35             low[u] = min(low[u], low[v]);
36         } else if (!scc[v]) {
37             low[u] = min(low[u], dfn[v]);
38         }
39     }
40     if (dfn[u] == low[u]) {
41         ++_scc;
42         do {
43             scc[stk[top]] = _scc;
44         } while (stk[top--] != u);
45     }
46 }
47 };

```

## 46.2 DFS

$O(nm)$  所求结果字典序最小

```

1 struct TWO_SAT {
2     int n, cnt;
3     int res[N], mem[N<<1], mark[N<<1];
4     vector<int> e[N<<1];
5     void init(const int &n) {
6         n = _n;
7         memset(mark, 0, sizeof(int)*n*2);
8         for (int i = 0; i < n<<1; ++i) vector<int>().swap(e[i]);
9     }
10    // if u then v
11    void add_edge(const int &u, const int &v) {
12        e[u].emplace_back(v);
13    }
14    // pt i ==> i<<1 && i<<1/1 ==> 0 && 1
15    void add_edge(const int &u, const int &uv, const int &v, const int
        &vv) {
16        e[u<<1|uv].emplace_back(v<<1|vv);
17    }
18    // tag 0 any 1 smallest
19    bool work() {
20        for (int i = 0; i < n; ++i) {

```



```

21     if (mark[i<<1] || mark[i<<1|1]) continue;
22     cnt = 0;
23     if (!dfs(i<<1)) {
24         while (cnt) mark[mem[cnt--]] = 0;
25         if (!dfs(i<<1|1)) return false;
26     }
27 }
28 for (int i = 0; i < n<<1; ++i) if (mark[i]) res[i>>1] = i&1;
29 return true;
30 }
31 bool dfs(const int &u) {
32     if (mark[u^1]) return false;
33     if (mark[u]) return true;
34     mark[mem[++cnt] = u] = 1;
35     for (int v : e[u]) if (!dfs(v)) return false;
36     return true;
37 }
38 };

```

## 47 虚树

```

1  vector<int> ve[N];
2  void virtual_tree_clear(const int &u = 1) {
3      for (const int &v : ve[u]) virtual_tree_clear(v);
4      ve[u].clear();
5  }
6
7  // return the root of virtual tree
8  int virtual_tree_build(int vset[], const int &k) {
9      static int stk[N], top;
10     // id ==> dfn rank, d ==> depth
11     int *id = hld.id, *d = hld.d;
12     sort(vset+1, vset+k+1, & {
13         return id[x] < id[y];
14     });
15     top = 0;
16     int x, z;
17     for (int i = 1; i <= k; ++i) {
18         if (top && (z = hld.lca(vset[i], stk[top])) != stk[top]) {
19             x = stk[top--];
20             while (top && d[stk[top]] > d[z]) {
21                 ve[stk[top]].emplace_back(x);
22                 x = stk[top--];
23             }
24             ve[z].emplace_back(x);
25             if (!top || stk[top] != z) stk[++top] = z;
26         }
27         stk[++top] = vset[i];
28     }
29     x = stk[top--];
30     while (top) {
31         ve[stk[top]].emplace_back(x);
32         x = stk[top--];
33     }
34     // if (x != 1) ve[1].emplace_back(x); // force root at 1

```

```

35 | return x;
36 | }

```

## 48 线段树优化建图

```

1  template <typename T>
2  struct SegmentTreeGarph {
3      struct TreeNode {
4          int l, r;
5          int ls, rs;
6      } tr[N*3];
7      vector<pair<int, T>> *e;
8      int tot, root[2];
9      // op [down, 0] [up, 1]
10     template <typename E>
11     void build(const int &n, E *_e) {
12         tot = n;
13         e = _e;
14         for (int i = 1; i <= n; ++i) tr[i].l = tr[i].r = i;
15         build(1, n, root[0], 0);
16         build(1, n, root[1], 1);
17     }
18     void build(const int &l, const int &r, int &i, const int &op) {
19         if (l == r) return i = l, void();
20         i = ++tot;
21         tr[i].l = l; tr[i].r = r;
22         int mid = (l+r)>>1;
23         build(l, mid, tr[i].ls, op);
24         build(mid+1, r, tr[i].rs, op);
25         e[op ? tr[i].ls : i].emplace_back(op ? i : tr[i].ls, 0);
26         e[op ? tr[i].rs : i].emplace_back(op ? i : tr[i].rs, 0);
27     }
28     void insert(const int &o, const int &l, const int &r, const T &w,
29               const int &op) {
30         if (l == r) e[op ? l : o].emplace_back(op ? o : l, w);
31         else insert(o, l, r, w, op, root[op]);
32     }
33     void insert(const int &o, const int &l, const int &r, const T &w,
34               const int &op, const int &i) {
35         if (tr[i].l >= l && tr[i].r <= r) {
36             e[op ? i : o].emplace_back(op ? o : i, w);
37             return;
38         }
39         int mid = (tr[i].l+tr[i].r)>>1;
40         if (l <= mid) insert(o, l, r, w, op, tr[i].ls);
41         if (r > mid) insert(o, l, r, w, op, tr[i].rs);
42     }
43 };

```

## 49 矩阵树定理 | Kirchhoff

解决一张图的生成树个数计数问题 (详情见 oi-wiki)

## Part VI

# 数论

### 50 快排

```

1 void quick_sort(int l, int r) {
2     if(l >= r) return;
3     swap(a[l], a[l+rand()%(r-l)]);
4     int i = l, j = r, mid = a[l];
5     while(i < j) {
6         while(i < j && a[j] >= mid) --j;
7         swap(a[i], a[j]);
8         while(i < j && a[i] < mid) ++i;
9         swap(a[i], a[j]);
10    }
11    quick_sort(l, i-1);
12    quick_sort(i+1, r);
13 }

```

### 51 求逆序对 (归并排序)

```

1 void merge_sort(int l, int r) {
2     if(l == r) return;
3     int mid = (l+r)>>1;
4     merge_sort(l, mid);
5     merge_sort(mid+1, r);
6     int i = l, j = mid+1, k = 1;
7     while(k <= r) {
8         if(j <= r && (i > mid || a[j] < a[i])) {
9             ans += mid-i+1;
10            b[k++] = a[j++];
11        }
12        else b[k++] = a[i++];
13    }
14    memcpy(a+l, b+l, sizeof(int)*(r-l+1));
15 }

```

### 52 线性基

```

1 template <typename T>
2 struct LinearBase {
3     int sz = sizeof(T)*8, zero;
4     T tot;
5     vector<T> b, rb, p;
6     LinearBase(){ init(); }
7     void init() {
8         tot = zero = 0;;
9         vector<T>(sz, 0).swap(b);
10        vector<T>().swap(rb);

```

```

11     vector<T>().swap(p);
12 }
13 template <typename TT>
14 void build(TT a[], const int &n) {
15     init();
16     for (int i = 1; i <= n; ++i) insert(a[i]);
17 }
18 void merge(const LinearBase xj) {
19     for (int i : xj.b) if (i) insert(i);
20 }
21 void insert(T x) {
22     for (int i = sz-1; i >= 0; --i) if ((x>>i)&1) {
23         if (!b[i]) { b[i] = x; return; }
24         x ^= b[i];
25     }
26     zero = 1;
27 }
28 bool find(T x) {
29     for (int i = sz-1; i >= 0; --i) if ((x>>i)&1) {
30         if (!b[i]) { return false; }
31         x ^= b[i];
32     }
33     return true;
34 }
35 T max_xor() {
36     T res = 0;
37     for (int i = sz-1; i >= 0; --i)
38         if (~(res>>i)&1) res ^= b[i];
39     // res = max(res, res^b[i]);
40     return res;
41 }
42 T min_xor() {
43     if (zero) return 0;
44     for (int i = 0; i < sz; ++i)
45         if (b[i]) return b[i];
46 }
47 void rebuild() {
48     rb = b;
49     vector<T>().swap(p);
50     for (int i = sz-1; i >= 0; --i)
51         for (int j = i-1; j >= 0; --j)
52             if ((rb[i]>>j)&1) rb[i] ^= rb[j];
53     for (int i = 0; i < sz; ++i)
54         if (rb[i]) p.emplace_back(rb[i]);
55     tot = ((T)1<<p.size())>zero;
56 }
57 T kth_min(T k) {
58     if (k >= tot || k < 1) return -1;
59     if (zero && k == 1) return 0;
60     if (zero) --k;
61     T res = 0;
62     for (int i = (int)p.size()-1; i >= 0; --i)
63         if ((k>>i)&1) res ^= p[i];
64     return res;
65 }
66 T kth_max(const T &k) {
67     return kth_min(tot-k);

```

```

68 }
69 };

```

前缀和线性基 vector 跑贼鸡儿慢

```

1  template <class T>
2  struct PreSumLB {
3      int tot, sz = sizeof(T)*8;
4      vector<T> b[N];
5      vector<int> p[N];
6      PreSumLB() { init(); }
7      void init() {
8          tot = 0;
9          vector<T>(sz, 0).swap(b[0]);
10         vector<int>(sz, 0).swap(p[0]);
11     }
12     void append(T val) {
13         int pos = ++tot;
14         vector<T> &bb = b[tot];
15         vector<int> &pp = p[tot];
16         pp = p[tot-1];
17         bb = b[tot-1];
18         for (int i = sz-1; i >= 0; --i) if ((val>>i)&1) {
19             if (bb[i]) {
20                 if (pos > pp[i]) swap(pos, pp[i]), swap(val, bb[i]);
21                 val ^= bb[i];
22             } else {
23                 bb[i] = val;
24                 pp[i] = pos;
25                 return;
26             }
27         }
28     }
29     T query(const int &l, const int &r) {
30         T res = 0;
31         vector<T> &bb = b[r];
32         vector<int> &pp = p[r];
33         for (int i = sz-1; i >= 0; --i)
34             if (pp[i] >= l) res = max(res, res^bb[i]);
35         return res;
36     }
37 };

```

## 53 矩阵

```

1  template <typename T>
2  struct Martix {
3      int n, m;
4      T a[N][N];
5      Martix(){}
6      Martix(const int &n) : n(_n), m(_n) { init(); }
7      Martix(const int &n, const int &m) : n(_n), m(_m) { init(); }
8      T* operator [] (const int &i) { return a[i]; }
9      void init(const int &>tag = 0) {
10         for (int i = 1; i <= n; ++i) memset(a[i], 0, sizeof(T)*(n+1));
11         for (int i = 1; i <= n; ++i) a[i][i] = tag;

```

```

12 }
13 friend Martix operator * (const Martix &m1, const Martix &m2) {
14     Martix res(m1.n, m2.m);
15     for (int i = 1; i <= res.n; ++i)
16         for (int j = 1; j <= res.m; ++j)
17             for (int k = 1; k <= m1.m; ++k)
18                 res.a[i][j] = (res.a[i][j]+m1.a[i][k]*m2.a[k][j])%MOD;
19     return res;
20 }
21 Martix& operator *= (const Martix &mx) { return *this = *this*mx; }
22 template <typename TT>
23 Martix pow(const TT &p) const {
24     Martix res(n, m), a = *this;
25     res.init(1);
26     for (TT i = p; i; i >>= 1, a *= a) if (i&1) res *= a;
27     return res;
28 }
29 Martix inv() const {
30     Martix res = *this;
31     vector<int> is(n+1), js(n+1);
32     for (int k = 1; k <= n; ++k) {
33         for (int i = k; i <= n; ++i)
34             for (int j = k; j <= n; ++j) if (res.a[i][j]) {
35                 is[k] = i; js[k] = j; break;
36             }
37         for (int i = 1; i <= n; ++i) swap(res.a[k][i], res.a[is[k]][i]);
38         for (int i = 1; i <= n; ++i) swap(res.a[i][k], res.a[i][js[k]]);
39         if (!res.a[k][k]) return Martix(0);
40         res.a[k][k] = mul_inverse(res.a[k][k]); // get inv of number
41         for (int j = 1; j <= n; ++j) if (j != k)
42             res.a[k][j] = res.a[k][j]*res.a[k][k]%MOD;
43         for (int i = 1; i <= n; ++i) if (i != k)
44             for (int j = 1; j <= n; ++j) if (j != k)
45                 res.a[i][j] = (res.a[i][j]+MOD-res.a[i][k]*res.a[k][j]%MOD)
46                     %MOD;
47         for (int i = 1; i <= n; ++i) if (i != k)
48             res.a[i][k] = (MOD-res.a[i][k]*res.a[k][k]%MOD)%MOD;
49     }
50     for (int k = n; k; --k) {
51         for (int i = 1; i <= n; ++i) swap(res.a[js[k]][i], res.a[k][i]);
52         for (int i = 1; i <= n; ++i) swap(res.a[i][is[k]], res.a[i][k]);
53     }
54     return res;
55 }
56 T det() {
57     long long res = 1;
58     Martix cpy = *this;
59     for (int i = 1; i <= n; ++i) {
60         for (int j = i+1; j <= n; ++j) while (cpy.a[j][i]) {
61             long long t = cpy.a[i][i]/cpy.a[j][i];
62             for (int k = i; k <= n; ++k)
63                 cpy.a[i][k] = (cpy.a[i][k]+MOD-t*cpy.a[j][k]%MOD)%MOD;
64             swap(cpy.a[i], cpy.a[j]);

```

```

64         res = -res;
65     }
66     res = res*cpy.a[i][i]%MOD;
67 }
68 return (res+MOD)%MOD;
69 }
70 friend ostream& operator << (ostream &os, Martix<T> &mx) {
71     for (int i = 1; i <= mx.n; ++i)
72         for (int j = 1; j <= mx.m; ++j)
73             os << mx[i][j] << " \n"[j==mx.m];
74     return os;
75 }
76 };

```

## 54 高斯消元

```

1 struct GaussElimination {
2     double a[N][N];
3     void init() { memset(a, 0, sizeof a); }
4     void init(const int &n) {
5         for (int i = 1; i <= n; ++i)
6             for (int j = 1; j <= n+1; ++j)
7                 a[i][j] = 0;
8     }
9     // ans is a[i][n+1]
10    bool solve(const int &n) {
11        for (int i = 1, j, k; i <= n; ++i) {
12            for (j = i+1, k = i; j <= n; ++j)
13                if (abs(a[j][i]) > abs(a[k][i])) k = j;
14            if (abs(a[k][i]) < eps) return false;
15            swap(a[k], a[i]);
16            for (j = 1; j <= n; ++j) if (i != j) {
17                double d = a[j][i]/a[i][i];
18                for (k = i+1; k <= n+1; ++k)
19                    a[j][k] -= d*a[i][k];
20            }
21        }
22        for (int i = 1; i <= n; ++i) a[i][n+1] /= a[i][i];
23        return true;
24    }
25 };

```

### 54.1 异或方程组

$a[i][j]$  第  $i$  个是否对  $j$  有影响  
 $a[i][n+1]$  第  $i$  个最后被翻转与否

```

1 // -1 : no solution, 0 : multi, 1 : one
2 template <typename T>
3 int XorGauss(T a[N], const int &n) {
4     for (int i = 1, j, k; i <= n; ++i) {
5         for (k = i; !a[k][i] && k <= n; ++k) {}
6         if (k <= n) swap(a[k], a[i]);
7         for (j = 1; j <= n; ++j) if (i != j && a[j][i])

```

```

8         for (k = i; k <= n+1; ++k) a[j][k] ^= a[i][k];
9         // a[j] ^= a[i]; // bitset<N> a[N]
10    }
11    for (int i = 1; i <= n; ++i) if (!a[i][i]) return -a[i][n+1];
12    return 1;
13 }
14 // dfs(n, 0)
15 void dfs(const int &u, const int &num) {
16     if (num >= res) return;
17     if (u <= 0) { res = num; return; }
18     if (a[u][u]) {
19         int t = a[u][n+1];
20         for (int i = u+1; i <= n; ++i) {
21             if (a[u][i]) t ^= used[i];
22         }
23         dfs(u-1, num+t);
24     } else { // 自由元
25         dfs(u-1, num);
26         used[u] = 1;
27         dfs(u-1, num+1);
28         used[u] = 0;
29     }
30 }

```

## 55 拉格朗日插值

```

1 template <typename T, typename H, typename P>
2 long long Largrange(const T &k, const int &n, const H x[], const P y
3     []) {
4     k %= MOD;
5     long long res = 0, s1 = 1, s2 = 1;
6     for (int i = 1; i <= n; ++i, s1 = s2 = 1) {
7         for (int j = 1; j <= n; ++j) if (i != j) {
8             s1 = s1*(x[i]-x[j]+MOD)%MOD;
9             s2 = s2*(k-x[j]+MOD)%MOD;
10        }
11        res = (res+y[i]*s2%MOD*mul_inverse(s1)%MOD)%MOD;
12    }
13    return res;
14 }

```

```

1 template <typename T, typename P> // x[i] = i -> y[i] = f(i)
2 long long Largrange(const T &k, const int &n, const P y[]) {
3     if (k <= n) return y[k];
4     static long long pre[N], suf[N];
5     long long res = 0;
6     k %= MOD;
7     pre[0] = suf[n+1] = 1;
8     for (int i = 1; i <= n; ++i) pre[i] = pre[i-1]*(k-i)%MOD;
9     for (int i = n; i >= 1; --i) suf[i] = suf[i+1]*(k-i)%MOD;
10    for (int i = 1; i <= n; ++i) {
11        res = (res+y[i]*(pre[i-1]*suf[i+1]%MOD)%MOD*
12            mul_inverse(((n-i)&1 ? -1 : 1)*fac[i-1]*fac[n-i]%MOD)%MOD)%MOD;
13    }
14 }

```



```

14 | return (res+MOD)%MOD;
15 | }

```

## 56 快速乘

```

1 | inline long long qmul(long long x, long long y, long long mo) {
2 |     long long res = 0;
3 |     while (y) {
4 |         if (y&1) res = (res+x)%mo;
5 |         x = (x<<1)%mo;
6 |         y >>= 1;
7 |     }
8 |     return res;
9 | }

```

```

1 | inline long long qmul(long long x, long long y, long long mo) {
2 |     return (long long)((__int128)x*y%mo);
3 | }

```

```

1 | inline long long qmul(long long x, long long y, long long mo) {
2 |     // x*y - floor(x*y/mo)*mo
3 |     typedef unsigned long long ull;
4 |     typedef long double ld;
5 |     return ((ull)x*y-(ull)((ld)x/mo*y)*mo+mo)%mo;
6 | }

```

## 57 复数

```

1 | struct comp {
2 |     typedef double T; // maybe Long double ?
3 |     T real, imag;
4 |     comp (const double &_real = 0, const double &_imag = 0) : real(
5 |         _real), imag(_imag) {}
6 |     friend comp operator + (const comp &c1, const comp &c2) { return
7 |         comp(c1.real+c2.real, c1.imag+c2.imag); }
8 |     friend comp operator - (const comp &c1, const comp &c2) { return
9 |         comp(c1.real-c2.real, c1.imag-c2.imag); }
10 |    friend comp operator * (const comp &c1, const comp &c2) { return
11 |        comp(c1.real*c2.real-c1.imag*c2.imag, c1.real*c2.imag+c1.imag*c2
12 |            .real); }
13 |    comp& operator += (const comp &c) { return *this = *this+c; }
14 |    comp& operator -= (const comp &c) { return *this = *this-c; }
15 |    comp& operator *= (const comp &c) { return *this = *this*c; }
16 |    friend istream& operator >> (istream &is, comp &c) { return is >> c
17 |        .real >> c.imag; }
18 |    friend ostream& operator << (ostream &os, comp &c) { return os << c
19 |        .real << setiosflags(ios::showpos) << c.imag << "i"; }
20 |    comp conjugate() { return comp(real, -imag); }
21 |    friend comp conjugate(const comp &c) { return comp(c.real, -c.imag)
22 |        ; }
23 | };

```

## 58 快速傅里叶变换 |FFT

```

1 // array [0, n)
2 namespace FFT {
3     static const int SIZE = (1<<18)+3;
4     int len, bit;
5     int rev[SIZE];
6     // #define comp complex<long double>
7     void fft(comp a[], int flag = 1) {
8         for (int i = 0; i < len; ++i)
9             if (i < rev[i]) swap(a[i], a[rev[i]]);
10        for (int base = 1; base < len; base <= 1) {
11            comp w, wn = {cos(PI/base), flag*sin(PI/base)};
12            for (int i = 0; i < len; i += base*2) {
13                w = { 1.0, 0.0 };
14                for (int j = 0; j < base; ++j) {
15                    comp x = a[i+j], y = w*a[i+j+base];
16                    a[i+j] = x+y;
17                    a[i+j+base] = x-y;
18                    w *= wn;
19                }
20            }
21        }
22    }
23    void work(comp f[], const int &n, comp g[], const int &m) {
24        len = 1; bit = 0;
25        while (len < n+m) len <= 1, ++bit;
26        // multi-testcase
27        for (int i = n; i < len; ++i) f[i] = 0;
28        for (int i = m; i < len; ++i) g[i] = 0;
29        for (int i = 0; i < len; ++i)
30            rev[i] = (rev[i]>>1)>>1|((i&1)<<(bit-1));
31        fft(f, 1); fft(g, 1);
32        for (int i = 0; i < len; ++i) f[i] *= g[i];
33        fft(f, -1);
34        for (int i = 0; i < n+m; ++i) f[i].real /= len;
35    }
36 }

```

## 59 快速数论变换 |NTT

```

1 // array [0, n)
2 namespace NTT {
3     static const int SIZE = (1<<18)+3;
4     const int G = 3;
5     int len, bit;
6     int rev[SIZE];
7     long long f[SIZE], g[SIZE];
8     template <class T>
9     void ntt(T a[], int flag = 1) {
10        for (int i = 0; i < len; ++i)
11            if (i < rev[i]) swap(a[i], a[rev[i]]);
12        for (int base = 1; base < len; base <= 1) {
13            long long wn = qpow(G, (MOD-1)/(base*2)), w;

```

```

14     if (flag == -1) wn = qpow(wn, MOD-2);
15     for (int i = 0; i < len; i += base*2) {
16         w = 1;
17         for (int j = 0; j < base; ++j) {
18             long long x = a[i+j], y = w*a[i+j+base]%MOD;
19             a[i+j] = (x+y)%MOD;
20             a[i+j+base] = (x-y+MOD)%MOD;
21             w = w*wn%MOD;
22         }
23     }
24 }
25 }
26 template <class T>
27 void work(T a[], const int &n, T b[], const int &m) {
28     len = 1; bit = 0;
29     while (len < n+m) len <<= 1, ++bit;
30     for (int i = 0; i < n; ++i) f[i] = a[i];
31     for (int i = n; i < len; ++i) f[i] = 0;
32     for (int i = 0; i < m; ++i) g[i] = b[i];
33     for (int i = m; i < len; ++i) g[i] = 0;
34     for (int i = 0; i < len; ++i)
35         rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
36     ntt(f, 1); ntt(g, 1);
37     for (int i = 0; i < len; ++i) f[i] = f[i]*g[i]%MOD;
38     ntt(f, -1);
39     long long inv = qpow(len, MOD-2);
40     for (int i = 0; i < n+m-1; ++i) f[i] = f[i]*inv%MOD;
41 }
42 }

```

## 60 任意模数 NTT|MTT

```

1 namespace MTT {
2     static const int SIZE = (1<<18)+7;
3     int Mod = MOD;
4     comp w[SIZE];
5     int bitrev[SIZE];
6     long long f[SIZE];
7     void fft(comp *a, const int &n) {
8         for (int i = 0; i < n; ++i) if (i < bitrev[i]) swap(a[i], a[
9             bitrev[i]]);
10        for (int i = 2, lyc = n >> 1; i <= n; i <<= 1, lyc >>= 1)
11            for (int j = 0; j < n; j += i) {
12                comp *l = a + j, *r = a + j + (i >> 1), *p = w;
13                for (int k = 0; k < i>>1; ++k) {
14                    comp tmp = *r * *p;
15                    *r = *l - tmp, *l = *l + tmp;
16                    ++l, ++r, p += lyc;
17                }
18            }
19    }
20    template <class T>
21    inline void work(T *x, const int &n, T *y, const int &m) {
22        static int bit, L;
23        static comp a[SIZE], b[SIZE];

```

```

23 static comp dfta[SIZE], dftb[SIZE];
24
25 for (L = 1, bit = 0; L < n+m-1; ++bit, L <= 1);
26 for (int i = 0; i < L; ++i)
27     bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (bit - 1));
28 for (int i = 0; i < L; ++i)
29     w[i] = comp(cos(2 * PI * i / L), sin(2 * PI * i / L));
30
31 for (int i = 0; i < n; ++i)
32     (x[i] += Mod) %= Mod, a[i] = comp(x[i] & 32767, x[i] >> 15);
33 for (int i = n; i < L; ++i) a[i] = 0;
34 for (int i = 0; i < m; ++i)
35     (y[i] += Mod) %= Mod, b[i] = comp(y[i] & 32767, y[i] >> 15);
36 for (int i = m; i < L; ++i) b[i] = 0;
37 fft(a, L), fft(b, L);
38 for (int i = 0; i < L; ++i) {
39     int j = (L - i) & (L - 1);
40     static comp da, db, dc, dd;
41     da = (a[i] + conjugate(a[j])) * comp(.5, 0);
42     db = (a[i] - conjugate(a[j])) * comp(0, -.5);
43     dc = (b[i] + conjugate(b[j])) * comp(.5, 0);
44     dd = (b[i] - conjugate(b[j])) * comp(0, -.5);
45     dfta[j] = da*dc + da*dd*comp(0, 1);
46     dftb[j] = db*dc + db*dd*comp(0, 1);
47 }
48 for (int i = 0; i < L; ++i) a[i] = dfta[i];
49 for (int i = 0; i < L; ++i) b[i] = dftb[i];
50 fft(a, L), fft(b, L);
51 for (int i = 0; i < L; ++i) {
52     int da = (long long)(a[i].real / L + 0.5) % Mod;
53     int db = (long long)(a[i].imag / L + 0.5) % Mod;
54     int dc = (long long)(b[i].real / L + 0.5) % Mod;
55     int dd = (long long)(b[i].imag / L + 0.5) % Mod;
56     f[i] = (da + ((long long)(db + dc) << 15)
57             + ((long long)dd << 30)) % Mod;
58 }
59 for (int i = 0; i < n+m-1; ++i) (f[i] += Mod) %= Mod;
60 }
61 }

```

## 61 分治 FFT

```

1 // give g[1, n) ask f[0, n)
2 // f[i] = sigma f[i-j]*g[j] (1 <= j <= i)
3 template <class T> // [l, r)
4 void cdq_fft(T f[], T g[], const int &l, const int &r) {
5     if (r-l <= 1) return;
6     int mid = (l+r)>>1;
7     cdq_fft(f, g, l, mid);
8     NTT::work(f+l, mid-l, g, r-l);
9     for (int i = mid; i < r; ++i)
10         (f[i] += NTT::f[i-l]) %= MOD;
11     cdq_fft(f, g, mid, r);
12 }

```

```
13 // f[0] = 1; cdq_fft(f, g, 0, n);
```

## 62 快速沃尔什变换 | FWT

复杂度  $O(n \log n) | O(n2^n)$

$$FWT(A \pm B) = FWT(A) \pm FWT(B)$$

$$FWT(cA) = cFWT(A)$$

定义  $\oplus$  为任意集合运算

$$FWT(A \oplus B) = FWT(A) \times FWT(B)$$

$$\text{求 } C_i = \sum_{i=j \oplus k} a_j b_k$$

### 62.1 或运算

$$FWT(A)[i] = \sum_{j|i=i} A[j]$$

$$FWT(A) = [FWT(A_0), FWT(A_0 + A_1)]$$

$$IFWT(A) = [IFWT(A_0), IFWT(A_1) - IFWT(A_0)]$$

### 62.2 与运算

$$FWT(A)[i] = \sum_{i \& j=j} A[j]$$

$$FWT(A) = [FWT(A_0 + A_1), FWT(A_1)]$$

$$IFWT(A) = [IFWT(A_0) - IFWT(A_1), IFWT(A_1)]$$

### 62.3 异或运算

令  $d(x)$  为  $x$  在二进制下拥有的 1 的数量

$$FWT(A)[i] = \sum_j (-1)^{d(i \& j)} A[j]$$

$$FWT(A) = [FWT(A_0 + A_1), FWT(A_0 - A_1)]$$

$$IFWT(A) = [\frac{IFWT(A_1 - A_0)}{2}, \frac{IFWT(A_1 + A_0)}{2}]$$

### 62.4 code

```
1 namespace FWT {
2 #define forforfor for (int l = 2; l <= len; l <= 1)\
3     for (int i = 0, k = l>>1; i < len; i += l)\
4     for (int j = 0; j < k; ++j)
5
6 const int SIZE = (1<<17)+3;
7 int len;
8 int f[SIZE], g[SIZE];
9 template <class T> void init(T a[], const int &n, T b[], const int
    &m) {
10     len = 1;
11     while (len < max(n, m)) len <= 1;
12     for (int i = 0; i < n; ++i) f[i] = a[i];
13     for (int i = n; i < len; ++i) f[i] = 0;
14     for (int i = 0; i < m; ++i) g[i] = b[i];
15     for (int i = m; i < len; ++i) g[i] = 0;
```

```

16 }
17 template <class T> void fwt_or(T a[], const int x = 1) {
18     forforfor a[i+j+k] = (a[i+j+k]+1ll*a[i+j]*x)%MOD;
19 }
20 template <class T> void fwt_and(T a[], const int x = 1) {
21     forforfor a[i+j] = (a[i+j]+1ll*a[i+j+k]*x)%MOD;
22 }
23 template <class T> void fwt_xor(T a[], const int x = 1) {
24     forforfor {
25         (a[i+j] += a[i+j+k]) %= MOD;
26         a[i+j+k] = (a[i+j]-2*a[i+j+k]%MOD+MOD)%MOD;
27         a[i+j] = 1ll*a[i+j]*x%MOD; a[i+j+k] = 1ll*a[i+j+k]*x%MOD;
28     }
29 }
30 template <class T> void work_or(const T a[], const int &n, const T
    b[], const int &m) {
31     init(a, n, b, m); fwt_or(f); fwt_or(g);
32     for (int i = 0; i < len; ++i) f[i] = 1ll*f[i]*g[i]%MOD;
33     fwt_or(f, MOD-1); // fwt_or(x, -1)
34 }
35 template <class T> void work_and(const T a[], const int &n, const T
    b[], const int &m) {
36     init(a, n, b, m); fwt_and(f); fwt_and(g);
37     for (int i = 0; i < len; ++i) f[i] = 1ll*f[i]*g[i]%MOD;
38     fwt_and(f, MOD-1); // fwt_and(x, -1)
39 }
40 template <class T> void work_xor(const T a[], const int &n, const T
    b[], const int &m) {
41     init(a, n, b, m); fwt_xor(f); fwt_xor(g);
42     for (int i = 0; i < len; ++i) f[i] = 1ll*f[i]*g[i]%MOD;
43     fwt_xor(f, mul_inverse(2)); // fwt_xor(x, 1/2)
44 }
45 #undef forforfor
46 } // namespace FWT

```

## 63 快速子集变换 (子集卷积)|FST

$$C_k = \sum_{i \oplus j = k} A_i B_j$$

复杂度  $O(n \log^2 n) | O(n^2 2^n)$

```

1 namespace FST {
2     const int W = 20;
3     const int N = 1<<W;
4     int len, bit;
5     int f[W+1][N], g[W+1][N], h[W+1][N], res[N];
6     template <class T> void fwt(T a[], const int x = 1) {
7         for (int l = 2; l <= len; l <<= 1)
8             for (int i = 0, k = l>>1; i < len; i += l)
9                 for (int j = 0; j < k; ++j)
10                     a[i+j+k] = (a[i+j+k]+1ll*a[i+j]*x)%MOD;
11     }
12     template <class T> void work(const T a[], const int &n, const T b
        [], const int &m) {
13         len = 1; bit = 0;

```

```

14 while (len < max(n, m)) len <= 1, ++bit;
15 for (int i = 0; i <= bit; ++i)
16     for (int j = 0; j < len; ++j)
17         f[i][j] = g[i][j] = h[i][j] = 0;
18 for (int i = 0; i < n; ++i) f[__builtin_popcount(i)][i] = a[i];
19 for (int i = 0; i < m; ++i) g[__builtin_popcount(i)][i] = b[i];
20 for (int i = 0; i <= bit; ++i) {
21     fwt(f[i]); fwt(g[i]);
22     for (int j = 0; j <= i; ++j)
23         for (int k = 0; k < len; ++k)
24             h[i][k] = (h[i][k] + 1ll * f[j][k] * g[i-j][k]) % MOD;
25     fwt(h[i], MOD-1); // fwt(h[i], -1)
26 }
27 for (int i = 0; i < len; ++i) res[i] = h[__builtin_popcount(i)][i];
28 }
29 } // namespace FST

```

### 63.1 倍增子集卷积

设多项式  $A = \sum_{i=0}^{2^n-1} a_i x^i, B = \sum_{i=0}^{2^n-1} b_i x^i$

求  $C = A * B = \sum_{i=0}^{2^n-1} x^i \sum_{d \subseteq i} a_d b_{i-d}$

按照每个状态的最高位进行分组，然后卷  $n$  次

复杂度  $O(\sum_{i=1}^n i^2 2^i) = O(n^2 2^n)$

```

1 template <typename T> void vip_fst(T a[], const int &n) { // return a
2     static int b[1<<B]; // warning: the type of b
3     int len = 1; while (len < n) len <= 1;
4     memcpy(b, a, sizeof(T)*len);
5     memset(a, 0, sizeof(T)*len); a[0] = 1;
6     for (int i = 1; i < len; i <= 1) {
7         FST::work(a, i, b+i, i);
8         for (int j = 0; j < i; ++j)
9             a[i+j] = FST::h[__builtin_popcount(j)][j];
10    }
11 }

```

## 64 第二类斯特林数

```

1 inline void stirling(const int &n) {
2     S[0][0] = 1;
3     for (int i = 1; i <= n; ++i)
4         for (int j = 1; j <= i; ++j)
5             S[i][j] = S[i-1][j-1] + S[i-1][j] * j;
6 }

```

```

1 void stirling(const int &n) {
2     inv[0] = inv[1] = 1;
3     for (int i = 2; i <= n; ++i)
4         inv[i] = MOD - MOD / i * inv[MOD % i] % MOD;

```

```

5   for (int i = 1; i <= n; ++i)
6       inv[i] = inv[i-1]*inv[i]%MOD;
7   while (len <= (n<<1)) len <= 1, ++bit;
8   for (int i = 0; i < len; ++i)
9       rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
10  for (int i = 0, one = 1; i <= n; ++i, one = MOD-one) {
11      f[i] = one*inv[i]%MOD;
12      g[i] = qpow(i, n)*inv[i]%MOD;
13  }
14  NTT(f, 1); NTT(g, 1);
15  for (int i = 0; i < len; ++i) f[i] = f[i]*g[i]%MOD;
16  NTT(f, -1);
17  long long invv = qpow(len, MOD-2);
18  for (int i = 0; i <= n; ++i)
19      printf("%lld%c", f[i]*invv%MOD, " \n"[i==n]);
20 }

```

## 65 约瑟夫环

$O(n)$

```

1 int solve(int n, int v) { return n == 1 ? 0 : (solve(n-1, v)+v)%n; }
2 // res = solve(num, step)+1

```

## 66 最小公倍数 lcm

$$LCM\left(\frac{a}{b}, \frac{c}{d}\right) = \frac{LCM(a, c)}{GCD(b, d)}$$

$$LCM\left(\frac{a_1}{b_1}, \frac{a_2}{b_2}, \dots\right) = \frac{LCM(a_1, a_2, \dots)}{GCD(b_1, b_2, \dots)}$$

## 67 扩展欧几里得（同余方程

```

1 template <typename T>
2 T exgcd(const T a, const T b, T &x, T &y) {
3     if (!b) return x = 1, y = 0, a;
4     T d = exgcd(b, a%b, y, x);
5     y -= a/b*x;
6     return d;
7 }

```

## 68 乘法逆元

### 68.1 拓展欧几里得

```

1 template <typename T>
2 inline T mul_inverse(const T &a, const T &mo = MOD) {
3     T x, y;
4     exgcd(a, mo, x, y);
5     return (x%mo+mo)%mo;
6 }

```



## 68.2 费马小定理

$$a^{p-1} \equiv 1(\text{mod } p) \Rightarrow \text{inv}(a) = a^{p-2}$$

## 68.3 线性递推

```

1 template <typename T>
2 inline void mul_inverse(T *inv, int mod = MOD) {
3     inv[0] = inv[1] = 1;
4     for(int i = 2; i <= n; ++i)
5         inv[i] = 1ll*(mod-mod/i)*inv[mod%i]%mod;
6 }

```

## 69 中国剩余定理

### 69.1 中国剩余定理 CRT(m 互质)

```

1 inline long long CRT(int a[], int m[]) {
2     long long res = 0, M = 1;
3     for (int i = 1; i <= n; ++i)
4         M *= m[i];
5     for (int i = 1; i <= n; ++i)
6         res = (res + a[i]*(M/m[i])*mul_inverse(M/m[i], m[i]))%M;
7     return (res+M)%M;
8 }

```

### 69.2 扩展中国剩余定理 EXCRT(m 不互质)

```

1 inline long long EXCRT(long long a[], long long m[]) {
2     // M*x + m[i]*y = a[i]-res (mod m[i])
3     // res = res+x*M;
4     long long M = m[1], res = a[1], x, y, c, d;
5     for (int i = 2; i <= n; ++i) {
6         d = exgcd(M, m[i], x, y);
7         c = (a[i]-res%m[i]+m[i])%m[i];
8         if (c%d != 0) return -1;
9         x = (c/d)*x%(m[i]/d);
10        res += x*M;
11        M *= m[i]/d;
12        res = (res%M+M)%M;
13    }
14    return res;
15 }

```

## 70 排列组合奇偶性

$C(n, k)$  当  $n \& k == k$  为奇数反之偶数

## 71 欧拉函数

```

1 template <typename T> inline T phi(T x) {
2     T res = x;
3     for (T i = 2; i*i <= x; ++i) {
4         if (x%i) continue;
5         res = res/i*(i-1);
6         while (x%i == 0) x /= i;
7     }
8     if (x > 1) res = res/x*(x-1);
9     return res;
10 }

```

### 71.1 筛法

```

1 struct Euler {
2     int phi[N], check[N];
3     vector<int> prime;
4     void init(int sz) {
5         for (int i = 1; i <= sz; ++i) check[i] = 1;
6         phi[1] = 1; check[1] = 0;
7         for (int i = 2; i <= sz; ++i) {
8             if (check[i]) {
9                 prime.emplace_back(i);
10                phi[i] = i-1;
11            }
12            for (int j : prime) {
13                if (i*j > sz) break;
14                check[i*j] = 0;
15                if (i%j) {
16                    phi[i*j] = (j-1)*phi[i];
17                } else {
18                    phi[i*j] = j*phi[i];
19                    break;
20                }
21            }
22        }
23    }
24 } E;

```

## 72 莫比乌斯函数

```

1 template <typename T> inline T miu(T x) {
2     int t = 0;
3     for (T i = 2, k; i*i <= x; ++i) {
4         if (x%i) continue;
5         for (k = 0, ++t; x %i == 0; x /= i, ++k) {}
6         if (k >= 2) return 0;
7     }
8     if (x > 1) ++t;
9     return t&1 ? -1 : 1;
10 }

```

## 73 线性筛素数

```

1 struct Euler {
2     int tot = 0;
3     int prime[N];
4     bool check[N];
5     bool& operator [] (const int i) { return check[i]; }
6     void init(int sz) {
7         tot = 0;
8         for (int i = 1; i <= sz; ++i) check[i] = true;
9         check[1] = false;
10        for (register int i = 2, j; i <= sz; ++i) {
11            if (check[i]) prime[++tot] = i;
12            for (j = 1; j <= tot && i*prime[j] <= sz; ++j) {
13                check[i*prime[j]] = false;
14                if (i%prime[j] == 0) break;
15            }
16        }
17    }
18 } E;

```

## 74 判断素数（质数）

### 74.1 Miller-Rabin 素性测试

```

1 inline bool MillerRabin(int x) {
2     static const int test_time = 10;
3     if (x < 3) return x == 2;
4     int a = x-1, b = 0;
5     while (!(a&1)) a >>= 1, ++b;
6     for (int i = 1, j, v; i <= test_time; ++i) {
7         v = (qpow(rnd()%(x-2)+2, a, x));
8         if (v == 1 || v == x-1) continue;
9         for (j = 0; j < b && v != x-1; ++j)
10            v = static_cast<int>(1ll*v*v%x);
11         if (j >= b) return false;
12     }
13     return true;
14 }

```

## 75 线性筛 GCD

```

1 inline void gcd_init(const int &n) {
2     for (int i = 1; i <= n; ++i)
3         for (int j = 1; j <= n; ++j) if (!g[i][j])
4             for (int k = 1; k <= n/i; ++k)
5                 g[k*i][k*j] = k;
6 }

```

## 76 BSGS

求解关于  $t$  的方程  $a^t \equiv x \pmod{m}, \gcd(a, m) = 1$

```

1 // map<long long, int> mmp; // a^n = x
2 inline long long BSGS(long long a, long long x, long long m) {
3     long long t = (long long)ceil(sqrt(m)); // b = a^i
4     for(int i = 0; i < t; ++i)
5         mmp[mul(x, qpow(a, i))] = i;
6     a = qpow(a, t);
7     long long now, ans; // now = (a^t)^i
8     for(int i = 0; i <= t; ++i) {
9         now = qpow(a, i);
10        if(mmp.count(now)) {
11            ans = t*i - mmp[now];
12            if(ans > 0) return ans;
13        }
14    }
15    return -1;
16 }

```

## 77 错排

$$\begin{aligned}
 D_1 &= 0 \\
 D_2 &= 1 \\
 D_n &= (n-1)(D_{n-1} + D_{n-2})
 \end{aligned}$$

## 78 原根

复杂度  $O(\sqrt{m} + g \times \log^2 m)$

```

1 inline int getG(const int &m) {
2     static int q[100000+7];
3     int _phi = phi(m), x = _phi, tot = 0;
4     for (int i = 2; i*i <= _phi; ++i) {
5         if (x%i) continue;
6         q[++tot] = _phi/i;
7         while (x%i == 0) x /= i;
8     }
9     if (x > 1) x = q[++tot] = _phi/x;
10    for (int g = 2, flag; ; ++g) {
11        flag = 1;
12        if (qpow(g, _phi, m) != 1) continue;
13        for (int i = 1; i <= tot; ++i) {
14            if (qpow(g, q[i], m) == 1) {
15                flag = 0;
16                break;
17            }
18        }
19        if (flag) return g;
20    }
21 }

```

## Part VII

# 动态规划 DP

## 79 线性 DP

### 79.1 最长公共子序列 LCS

```

1 f[i][j] = max{ f[i-1][j],
2               f[i][j-1],
3               f[i-1][j-1]+1 (if A[i] == B[j]) }

```

## 80 状压 DP

### 80.1 枚举子集

```

1 for (int i = s; i; i = (i-1)&s) {}

```

### 80.2 枚举 n 个元素大小为 k 的二进制子集

```

1 int s=(1<<k)-1;
2 while(s<(1<<n)){
3     work(s);
4     int x=s&-s,y=s+x;
5     s=((s&~y)/x>>1)|y; //这里有一个位反~
6 }

```

## 81 背包问题

### 81.1 多重背包

二进制拆分

```

1 for(int i = 1, cnt, vi, wi, m; i <= n; ++i) {
2     scanf("%d%d%d", &vi, &wi, &m);
3     cnt = 1;
4     while(m-cnt > 0) {
5         m -= cnt;
6         v.push_back(vi*cnt);
7         w.push_back(wi*cnt);
8         cnt <<= 1;
9     }
10    v.push_back(vi*m);
11    w.push_back(wi*m);
12 }
13 for(int i = 0; i < w.size(); ++i)
14     for(int j = W; j >= w[i]; --j)
15         b[j] = max(b[j], b[j-w[i]]+v[i]);

```

单调队列

```

1 for(int i = 1; i <= n; ++i) {
2     scanf("%d%d%d", &v, &w, &m);
3     for(int u = 0; u < w; ++u) {
4         int maxp = (W-u)/w;
5         head = 1; tail = 0;
6         for(int k = maxp-1; k >= max(0, maxp-m); --k) {
7             while(head <= tail && calc(u, q[tail]) <= calc(u, k)) tail--;
8             q[++tail] = k;
9         }
10        for(int p = maxp; p >= 0; --p) {
11            while(head <= tail && q[head] >= p) head++;
12            if(head <= tail) f[u+p*w] = max(f[u+p*w], p*v+calc(u, q[head]));
13            if(p-m-1 < 0) continue;
14            while(head <= tail && calc(u, q[tail]) <= calc(u, p-m-1)) tail--;
15            q[++tail] = p-m-1;
16        }
17    }
18 }
19 int ans = 0;
20 for(int i = 1; i <= W; ++i) ans = max(ans, f[i]);

```

## 82 斜率优化

若  $dp$  方程为  $dp[i] = a[i] \cdot b[j] + c[i] + d[j]$  时, 由于存在  $a[i] \cdot b[j]$  这个既有  $i$  又有  $j$  的项, 就需要使用斜率优化

### 82.1 「HNOI2008」玩具装箱 TOY

$$dp[i] = \min(dp[j] + (sum[i] + i - sum[j] - j - L - 1)^2) (j < i)$$

$$\text{令 } a[i] = sum[i] + i, b[i] = sum[i] + i + L + 1$$

$$dp[i] = dp[j] + (a[i] - b[j])^2$$

$$dp[i] = dp[j] + a[i]^2 - 2 \cdot a[i] \cdot b[j] + b[j]^2$$

$$2 \cdot a[i] \cdot b[j] + dp[i] - a[i]^2 = dp[j] + b[j]^2$$

将  $b[j]$  看作  $x$ ,  $dp[j] + b[j]^2$  看作  $y$ , 这个式子就可以看作一条斜率为  $2 \cdot a[i]$  的直线

而对于每个  $i$  来说,  $a[i]$  都是确定的, 类似线性规划

$dp[i]$  的含义转化为: 当上述直线过点  $P(b[j], dp[j] + b[j]^2)$  时, 直线在  $y$  轴的截距加上  $a[i]^2$  (一个定值) 而题目即为找这个截距的最小值

## 83 四边形不等式

### 83.1 2D1D

$$f_{l,r} = \min_{k=l}^{r-1} \{f_{l,k} + f_{k+1,r}\} + w(l,r) \quad (1 \leq l \leq r \leq n)$$

当  $w(l,r)$  满足特定性质

- 区间包含单调性: 如果对于任意  $l \leq l' \leq r' \leq r$ , 均有  $w(l',r') \leq w(l,r)$  成立, 则称函数  $w$  对于区间包含关系具有单调性。

- 四边形不等式：如果对于任意  $l_1 \leq l_2 \leq r_1 \leq r_2$ ，均有  $w(l_1, r_1) + w(l_2, r_2) \leq w(l_1, r_2) + w(l_2, r_1)$  成立，则称函数  $w$  满足四边形不等式（简记为“交叉小于包含”）。若等号永远成立，则称函数  $w$  满足四边形恒等式。

> 引理 1：若满足关于区间包含的单调性的函数  $w(l, r)$  满足四边形不等式，则状态  $f_{l,r}$  也满足四边形不等式。

> 定理 1：若状态  $f$  满足四边形不等式，记  $m_{l,r} = \min\{k : f_{l,r} = g_{k,l,r}\}$  表示最优决策点，则有  $m_{l,r-1} \leq m_{l,r} \leq m_{l+1,r}$

## 83.2 1D1D

$$f_r = \min_{l=1}^{r-1} \{f_l + w(l, r)\} \quad (1 \leq r \leq n)$$

> 定理 2：若函数  $w(l, r)$  满足四边形不等式，记  $h_{l,r} = f_l + w(l, r)$  表示从  $l$  转移过来的状态  $r$ ， $k_r = \min\{l | f_r = h_{l,r}\}$  表示最优决策点，则有  $\forall r_1 \leq r_2 : k_{r_1} \leq k_{r_2}$

```

1 void DP(int l, int r, int k_l, int k_r) {
2     int mid = (l + r) / 2, k = k_l;
3     // 求状态f[mid]的最优决策点
4     for (int i = k_l; i <= min(k_r, mid - 1); ++i)
5         if (w(i, mid) < w(k, mid)) k = i;
6     f[mid] = w(k, mid);
7     // 根据决策单调性得出左右两部分的决策区间，递归处理
8     if (l < mid) DP(l, mid - 1, k_l, k);
9     if (r > mid) DP(mid + 1, r, k, k_r);
10 }

```

## 83.3 满足四边形不等式的函数类

- 性质 1：若函数  $w_1(l, r), w_2(l, r)$  均满足四边形不等式（或区间包含单调性），则对于任意  $c_1, c_2 \geq 0$ ，函数  $c_1 w_1 + c_2 w_2$  也满足四边形不等式（或区间包含单调性）。

- 性质 2：若存在函数  $f(x), g(x)$  使得  $w(l, r) = f(r) - g(l)$ ，则函数  $w$  满足四边形恒等式。当函数  $f, g$  单调增加时，函数  $w$  还满足区间包含单调性。

- 性质 3：设  $h(x)$  是一个单调增加的凸函数，若函数  $w(l, r)$  满足四边形不等式并且对区间包含关系具有单调性，则复合函数  $h(w(l, r))$  也满足四边形不等式和区间包含单调性。

- 性质 4：设  $h(x)$  是一个凸函数，若函数  $w(l, r)$  满足四边形恒等式并且对区间包含关系具有单调性，则复合函数  $h(w(l, r))$  也满足四边形不等式。

首先需要澄清一点，凸函数（Convex Function）的定义在国内教材中有分歧，此处的凸函数指的是（可微的）下凸函数，即一阶导数单调增加的函数。

## 84 插头 DP | 轮廓线 DP

### 84.1 一个闭合回路

```

1 const int P = 299987;
2 const int M = 1<<21;
3 const int N = 15;
4
5 int n, m;
6 int a[N][N];
7 long long dp[2][M];
8 int head[2][P], nex[2][M], tot[2], ver[2][M];
9

```

```

10 inline void clear(const int &u) {
11     for (int i = 1; i <= tot[u]; ++i) {
12         dp[u][i] = 0; //
13         nex[u][i] = 0; //
14         head[u][ver[u][i]%P] = 0;
15     }
16     tot[u] = 0;
17 }
18
19 template <typename T, typename U>
20 inline void insert(const int &u, const T &x, const U &v) {
21     int p = x%P;
22     for (int i = head[u][p]; i; i = nex[u][i]) {
23         if (ver[u][i] == x) return dp[u][i] += v, void();
24     }
25     ++tot[u]; assert(tot[u] < M);
26     ver[u][tot[u]] = x;
27     nex[u][tot[u]] = head[u][p];
28     head[u][p] = tot[u];
29     dp[u][tot[u]] = v;
30 }
31
32 template <typename T>
33 inline int get_val(const int &u, const T &x) {
34     int p = x%P;
35     for (int i = head[u][p]; i; i = nex[u][i]) {
36         if (ver[u][i] == x) return dp[u][i];
37     }
38     return 0;
39 }
40
41 inline long long solve() {
42     int u = 0, base = (1<<m*2+2)-1;
43     long long res = 0;
44     clear(u);
45     insert(u, 0, 1);
46     for (int i = 1; i <= n; ++i) {
47         for (int j = 1; j <= m; ++j) {
48             clear(u ^= 1);
49             for (int k = 1; k <= tot[u^1]; ++k) {
50                 int state = ver[u^1][k];
51                 long long val = dp[u^1][k];
52                 if (j == 1) state = (state<<2)&base;
53                 // b1 right b2 down
54                 // 0 no 1 left 2 right
55                 int b1 = (state>>j*2-2)%4, b2 = (state>>j*2)%4;
56                 if (!a[i][j]) {
57                     if (!b1 && !b2) insert(u, state, val);
58                 } else if (!b1 && !b2) {
59                     if (a[i+1][j] && a[i][j+1]) insert(u, state+(1<<j*2-2)+(2<<
60                         j*2), val);
61                 } else if (!b1 && b2) {
62                     if (a[i][j+1]) insert(u, state, val);
63                     if (a[i+1][j]) insert(u, state+(b2<<j*2-2)-(b2<<j*2), val);
64                 } else if (b1 && !b2) {
65                     if (a[i+1][j]) insert(u, state, val);
66                     if (a[i][j+1]) insert(u, state-(b1<<j*2-2)+(b1<<j*2), val);

```



```

66     } else if (b1 == 1 && b2 == 1) { // find 2 turn to 1
67         for (int k = j+1, t = 1; k <= m; ++k) {
68             if ((state>>k*2)%4 == 1) ++t;
69             if ((state>>k*2)%4 == 2) --t;
70             if (!t) { insert(u, state-(1<<j*2-2)-(1<<j*2)-(1<<k*2),
71                             val); break; }
72         }
73     } else if (b1 == 2 && b2 == 2) { // find 1 turn to 2
74         for (int k = j-2, t = 1; k >= 0; --k) {
75             if ((state>>k*2)%4 == 1) --t;
76             if ((state>>k*2)%4 == 2) ++t;
77             if (!t) { insert(u, state-(2<<j*2-2)-(2<<j*2)+(1<<k*2),
78                             val); break; }
79         }
80     } else if (b1 == 2 && b2 == 1) {
81         insert(u, state-(2<<j*2-2)-(1<<j*2), val);
82     } else if (i == ex && j == ey) { // b1 == 1, b2 == 2
83         res += val;
84     }
85 }
86 return res;
87 }

```

## 84.2 多个闭合回路

```

1     else if (b1 == 1 && b2 == 2) {
2         if (i == ex && j == ey) res += val;
3         else dp[u][bit-(1<<j*2-2)-(1<<j*2+1)] += val;
4     }

```

## 84.3 联通块

```

1  int n, u, res = -INF;
2  int a[N][N];
3  unordered_map<int, int> dp[2];
4
5  inline void decode(const int &state, int *const s) {
6      for (int i = 1; i <= n; ++i) s[i] = (state>>i*3-3)%8;
7  }
8
9  inline void insert(const int *const s, const int &val) {
10     static int vis[N];
11     int state = 0, cnt = 0;
12     memset(vis, 0, sizeof vis);
13     for (int i = 1; i <= n; ++i) {
14         if (s[i] && !vis[s[i]]) vis[s[i]] = ++cnt;
15         state |= (vis[s[i]]<<i*3-3);
16     }
17     if (dp[u].count(state)) dp[u][state] = max(dp[u][state], val);
18     else dp[u].insert({state, val});
19     if (cnt == 1) res = max(res, val);

```

```

20 }
21
22 inline void solve() {
23     static int s[N];
24     dp[u = 0].clear();
25     dp[u][0] = 0;
26     for (int i = 1; i <= n; ++i) {
27         for (int j = 1; j <= n; ++j) {
28             dp[u ^ 1].clear();
29             for (const auto &p : dp[u^1]) {
30                 decode(p.first, s);
31                 int b1 = s[j-1], b2 = s[j];
32                 // not choose
33                 s[j] = 0;
34                 int cnt = 0;
35                 for (int k = 1; k <= n; ++k) cnt += s[k] == b2;
36                 if (!b2 || cnt) insert(s, p.second);
37                 s[j] = b2;
38                 // choose
39                 if (!b1 && !b2) {
40                     s[j] = 7;
41                 } else {
42                     if (b1 > b2) swap(b1, b2); // in case b2 == 0
43                     s[j] = b2;
44                     if (b1) for (int k = 1; k <= n; ++k) if (s[k] == b1) s[k] = b2;
45                 }
46                 insert(s, p.second+a[i][j]);
47             }
48         }
49     }
50     cout << res << endl;
51 }

```

## 84.4 L 型

L 型地板：拐弯且仅拐弯一次。

发现没有，一个存在的插头只有两种状态：拐弯过和没拐弯过，因此我们这样定义插头：

0 表没有插头，1 表没拐过的插头，2 表已经拐过的插头。b1 代表当前点的右插头，b2 代表当前点的下插头

## Part VIII

## STL

## 85 unordered\_map 重载

```

1 struct Node {
2     int a, b;
3     friend bool operator == (const Node &x, const Node &y) {
4         return x.a == y.a && x.b == y.b;
5     }
6 };

```

```
7 // 方法一
8 namespace std {
9     template <>
10     struct hash<Node> {
11         size_t operator () (const Node &x) const {
12             return hash<int>()(x.a)^hash<int>()(x.b);
13         }
14     };
15 }
16 unordered_map<Node, int> mp;
17 // 方法二
18 struct KeyHasher {
19     size_t operator () (const Node &x) const {
20         return hash<int>()(x.a)^hash<int>()(x.b);
21     }
22 };
23 unordered_map<Node, int, KeyHasher> mmp;
```