

## 目录

第一部分 杂项	5
1 快读快写	5
2 正则表达式	5
3 随机数	5
4 计算 $\log_2$	5
5 快速开根号   牛顿迭代法	6
6 $i/k == j$ 的 $k$ 的个数	6
7 三分法	6
第二部分 计算几何	6
8 向量坐标直线圆 (结构体)	6
9 二维凸包	8
10 平面最近点对	9
11 最小圆覆盖   随即增量法	10
第三部分 数据结构	10
12 堆	10
13 二叉查找树	10
14 平衡树	10
14.1 Splay	10
15 线段树	13
15.1 区间加减区间和	13
15.2 区间加减区间最值	15
16 ZKW 线段树	16
17 树状数组	17
17.1 一维	17
17.2 二维	18
17.2.1 单点修改区间查询	18
18 可持久化线段树 (主席树)	18
19 分块	19
20 ST 表	20
20.1 一维	20
20.2 二维	20
20.3 反向 ST	21
21 并查集	22
第四部分 字符串	22

<b>22 回文字符串   manacher 算法</b>	<b>22</b>
22.1 判断 $s[l, r]$ 是否为回文 . . . . .	23
<b>23 KMP</b>	<b>23</b>
<b>24 扩展 KMP Z 函数</b>	<b>23</b>
<b>25 字符串哈希</b>	<b>24</b>
<b>26 后缀数组  SA</b>	<b>24</b>
26.1 $O(n\log^2 n)$ . . . . .	24
26.2 $O(n\log n)$ . . . . .	25
26.3 $O(n)$ . . . . .	25
<b>27 字典树</b>	<b>26</b>
<b>28 AC 自动机</b>	<b>27</b>
 <b>第五部分 图论   树论</b>	 <b>29</b>
<b>29 DFS 树</b>	<b>29</b>
<b>30 树的重心</b>	<b>29</b>
<b>31 最大团</b>	<b>29</b>
<b>32 稳定婚姻匹配</b>	<b>30</b>
<b>33 最小生成树</b>	<b>31</b>
<b>34 二分图</b>	<b>31</b>
34.1 二分图匹配 . . . . .	31
34.2 二分图最小顶点覆盖 . . . . .	31
34.3 最大独立集 . . . . .	31
<b>35 LCA</b>	<b>31</b>
<b>36 树上差分</b>	<b>32</b>
<b>37 树链剖分</b>	<b>33</b>
<b>38 网络流</b>	<b>34</b>
38.1 最大流 . . . . .	34
38.1.1 Dinic . . . . .	34
38.1.2 ISAP . . . . .	35
38.1.3 HLPP . . . . .	36
38.2 最小割 . . . . .	36
38.3 费用流 . . . . .	36
38.3.1 ZKW SPFA . . . . .	36
38.4 上下界网络流 . . . . .	38
<b>39 最短路</b>	<b>38</b>
39.1 Floyd . . . . .	38
39.2 Dijkstra . . . . .	38
39.3 SPFA . . . . .	38
<b>40 负环</b>	<b>38</b>
<b>41 割点</b>	<b>38</b>
<b>42 SCC 强连通分量  Tarjan</b>	<b>39</b>
42.1 递归版本 . . . . .	39
<b>43 缩点</b>	<b>39</b>

<b>44 2-SAT</b>	<b>40</b>
44.1 SCC Tarjan	40
44.2 DFS	41
 <b>第六部分 数论</b>	 <b>41</b>
<b>45 快排</b>	<b>41</b>
<b>46 求第 K 大数</b>	<b>42</b>
<b>47 求逆序对 (归并排序)</b>	<b>42</b>
<b>48 线性基</b>	<b>42</b>
<b>49 矩阵</b>	<b>44</b>
49.1 矩阵快速幂	44
49.2 矩阵求逆	44
<b>50 高斯消元</b>	<b>45</b>
50.1 异或方程组	46
<b>51 拉格朗日插值</b>	<b>46</b>
<b>52 快速幂</b>	<b>47</b>
<b>53 快速乘</b>	<b>47</b>
<b>54 复数</b>	<b>48</b>
<b>55 快速傅里叶变换  FFT</b>	<b>48</b>
<b>56 快速数论变换  NTT</b>	<b>49</b>
<b>57 任意模数 NTT MTT</b>	<b>50</b>
<b>58 分治 FFT</b>	<b>51</b>
<b>59 第二类斯特林数</b>	<b>51</b>
<b>60 约瑟夫环</b>	<b>51</b>
60.1 $O(n)$	51
<b>61 最大公因数 gcd</b>	<b>52</b>
<b>62 最小公倍数 lcm</b>	<b>52</b>
<b>63 扩展欧几里得 (同余方程)</b>	<b>52</b>
<b>64 乘法逆元</b>	<b>52</b>
64.1 拓展欧几里得	52
64.2 费马小定理	52
64.3 线性递推	52
<b>65 中国剩余定理</b>	<b>53</b>
65.1 中国剩余定理 CRT(m 互质)	53
65.2 扩展中国剩余定理 EXCRT(m 不互质)	53
<b>66 排列组合</b>	<b>53</b>
66.1 奇偶性	53
<b>67 欧拉函数</b>	<b>53</b>
67.1 筛法	53
<b>68 线性筛</b>	<b>54</b>

<b>69 判断素数 (质数)</b>	<b>54</b>
69.1 Miller-Rabin 素性测试 . . . . .	54
<b>70 BSGS</b>	<b>55</b>
 <b>第七部分 动态规划 DP</b>	 <b>55</b>
<b>71 线性 DP</b>	<b>55</b>
71.1 最长上升子序列 LIS . . . . .	55
71.2 最长公共子序列 LCS . . . . .	55
71.3 数字三角形 . . . . .	56
<b>72 状压 DP</b>	<b>56</b>
72.1 枚举子集 . . . . .	56
72.2 枚举 $n$ 个元素大小为 $k$ 的二进制子集 . . . . .	56
<b>73 背包问题</b>	<b>56</b>
73.1 多重背包 . . . . .	56
 <b>第八部分 STL</b>	 <b>57</b>
<b>74 unordered_map 重载</b>	<b>57</b>
<b>75 定义函数</b>	<b>57</b>

## 第一部分 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 正则表达式

```

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

```

### 3 随机数

```

1 #include <random>
2 // 范围 unsigned int
3 mt19937 rnd(time(NULL));
4 mt19937 rnd(chrono::high_resolution_clock::now().time_since_epoch().count());
5 cout << rnd() << endl;

```

```

1 std::random_device rd; //获取随机数种子
2 std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
3 std::uniform_int_distribution<> dis(0, 9);
4 std::cout << dis(gen) << endl;

```

### 4 计算 log2

```

1 // lg2[i] = lg2(i) +1
2 for(int i = 1; i <= n; ++i) lg2[i] = lg2[i>>1]+1;
3 // lg2[i] = (int)log2(i)
4 for(int i = 2; i <= n; ++i) lg2[i] = lg2[i>>1]+1;

```

## 5 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 }

```

## 6 $i/k == j$ 的 $k$ 的个数

```

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)) {
10             cout << i << " " << j << endl;
11         }
12     }
13 }

```

## 7 三分法

```

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 }

```

```

1 while(r-l>5){
2     int ml=(l+l+r)/3;
3     int mr=(l+r+r)/3;
4     if(f(ml)<f(mr))r=mr;
5     else l=ml;
6 }
7 for (int i = l; i <= r; ++i) res = min(res, f(i));

```

```

1 while (r-l > 3) {
2     int mid = (l+r)>>1;
3     if (f(mid) < f(mid+1)) r = mid+1;
4     else l = mid;
5 }

```

## 第二部分 计算几何

## 8 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), id(_i) {}
7     friend Point operator + (const Point &p1, const Point &p2) {
8         return Point(p1.x+p2.x, p1.y+p2.y, p1.id);
9     }
10    friend Point operator - (const Point &p1, const Point &p2) {
11        return Point(p1.x-p2.x, p1.y-p2.y, p1.id);
12    }
13    friend Point operator - (const Point &p) {
14        return Point(-p.x, -p.y, p.id);
15    }
16    friend T operator * (const Point &p1, const Point &p2) {
17        return p1.x*p2.y-p1.y*p2.x;
18    }
19    template <typename TT>
20    friend Point operator / (const Point &p, const TT &k) {
21        return Point(p.x/k, p.y/k, p.id);
22    }
23    template <typename TT>
24    friend Point operator * (const Point &p, const TT &k) {
25        return Point(p.x*k, p.y*k, p.id);
26    }
27    Point operator += (const Point &p) { return *this = *this+p; }
28    Point operator -= (const Point &p) { return *this = *this-p; }
29    template <typename TT>
30    Point operator *= (const TT &k) { return *this = *this*k; }
31    template <typename TT>
32    Point operator /= (const TT &k) { return *this = *this/k; }
33    friend bool operator < (const Point &p1, const Point &p2) {
34        return make_pair(p1.x, p1.y) < make_pair(p2.x, p2.y);
35    }
36    friend bool operator > (const Point &p1, const Point &p2) {
37        return make_pair(p1.x, p1.y) > make_pair(p2.x, p2.y);
38    }
39    friend bool operator == (const Point &p1, const Point &p2) {
40        return p1.x == p2.x && p1.y == p2.y;
41    }
42    friend bool operator != (const Point &p1, const Point &p2) {
43        return p1.x != p2.x || p1.y != p2.y;
44    }
45    friend istream& operator >>(istream &is, Point &p) {
46        return is >> p.x >> p.y;
47    }
48    friend ostream& operator << (ostream &os, Point &p) {
49        return os << p.x << " " << p.y << " " << p.id << endl;
50    }
51    double length() { return sqrt(1.0*x*x+1.0*y*y); }
52    friend double dis(const Point &p1, const Point &p2) { return (p2-p1).length(); }
53    double dis(const Point &p) { return (p-*this).length(); }
54    friend T dot(const Point &p1, const Point &p2) { return p1.x*p2.x+p1.y*p2.y; }
55    T dot(const Point &p) { return x*p.x+y*p.y; }
56    friend Point rotate_90_c(const Point &p) { return Point(p.y, -p.x, p.id); }
57    Point rotate_90_c() { return Point(y, -x, id); }
58    friend double atan(const Point &p) { return atan2(p.y, p.x); }
59 };
60
61 template <typename T = double>
62 struct Vec { // 三维向量
63     T x, y, z;

```

```

64 Vec(const T &x = 0, const T &y = 0, const T &z = 0) : x(_x), y(_y), z(_z)
65 {}
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.x+v2.x,
68     v1.y+v2.y, v1.z+v2.z); }
69 friend Vec operator -(const Vec &v1, const Vec &v2) { return Vec(v1.x-v2.x,
70     v1.y-v2.y, v1.z-v2.z); }
71 friend Vec operator *(const T &k, const Vec &v) { return Vec(k*v.x, k*v.y, k*
72     v.z); }
73 friend Vec operator *(const Vec &v, const T &k) { return k*v; }
74 friend Vec operator *(const Vec &v1, const Vec &v2) {
75     return Vec(
76         v1.y*v2.z-v1.z*v2.y,
77         v1.z*v2.x-v1.x*v2.z,
78         v1.x*v2.y-v1.y*v2.x
79     );
80 }
81 friend T dot(const Vec &v1, const Vec &v2) { return v1.x*v2.x+v1.y*v2.y+v1.z*
82     v2.z; }
83 T dot(const Vec &v) { return dot(*this, v); }
84 Vec& operator +=(const Vec &v) { return *this = *this+v; }
85 Vec& operator -=(const Vec &v) { return *this = *this-v; }
86 Vec& operator *=(const T &k) { return *this = *this*k; }
87 Vec& operator *=(const Vec &v) { return *this = *this*v; }
88 friend istream& operator >>(istream &is, Vec &v) { return is >> v.x >> v.y >>
89     v.z; }
90 };
91
92 inline bool polar_angle1(const Point &p1, const Point &p2) {
93     double d1 = atan(p1), d2 = atan(p2);
94     return d1 == d2 ? p1 < p2 : d1 < d2;
95 }
96
97 inline bool polar_angle2(const Point &p1, const Point &p2) {
98     auto tmp = p1*p2;
99     return tmp == 0 ? p1 < p2 : tmp > 0;
100 }
101
102 inline long long S(const Point &p1, const Point &p2, const Point &p3) {
103     return abs(p1.x*p2.y+p1.y*p3.x+p2.x*p3.y-p1.x*p3.y-p1.y*p2.x-p2.y*p3.x);
104 }
105
106 struct Line {
107     Point p1, p2;
108     Line(){}
109     Line(const Point &p1, const Point &p2) : p1(_p1), p2(_p2) {}
110     friend bool cross(const Line &l1, const Line &l2) {
111         #define SJ1(x) max(l1.p1.x, l1.p2.x) < min(l2.p1.x, l2.p2.x) || \
112             max(l2.p1.x, l2.p2.x) < min(l1.p1.x, l1.p2.x)
113         if (SJ1(x) || SJ1(y)) return false;
114         #undef SJ1
115         #define SJ2(a, b, c, d) ((a-b)*(a-c))*((a-b)*(a-d)) <= 0
116         return SJ2(l1.p1, l1.p2, l2.p1, l2.p2) &&
117             SJ2(l2.p1, l2.p2, l1.p1, l1.p2);
118         #undef SJ2
119     }
120     friend bool on_line(const Line &l, const Point &p) {
121         return abs((l.p1-l.p2)*(l.p1-p)) < err;
122     }
123     friend Point cross_point(const Line &l1, const Line &l2) {
124         Point v1 = l1.p2-l1.p1, v2 = l2.p2-l2.p1;
125         if (abs(v1*v2) < err) return Point(0, 0); // no cross_point
126         double t = (l2.p1-l1.p1)*v2/(v1*v2);
127         return l1.p1+v1*t;
128     }
129 };

```



```

123 };
124
125 struct Circular {
126     Point o;
127     double r;
128     Circular(){}
129     Circular(const Point &o, const double &r) : o(o), r(r) {}
130     template <typename T>
131     Circular(const T &x, const T &y, const double &r) : o(Point(_x, _y)), r(_r) {}
132     friend bool in_cir(const Circular &c, const Point &p) { return dis(c.o, p) <= c.r; }
133     bool in_cir(const Point &p) { return dis(o, p) <= r; }
134 };
135
136 inline Circular get_cir(const Point &p1, const Point &p2, const Point &p3) {
137     Circular res;
138     res.o = cross_point(Line((p1+p2)/2, (p1+p2)/2+(p2-p1).rotate_90_c()),
139                       Line((p1+p3)/2, (p1+p3)/2+(p3-p1).rotate_90_c()));
140     res.r = dis(res.o, p1);
141     return res;
142 }

```

## 9 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 }

```

## 10 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  }

```

## 11 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 }

```

## 第三部分 数据结构

### 12 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 };

```

## 13 二叉查找树

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

```

```

67     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 线段树

### 15.1 区间加减区间和

```

1  template <typename T>
2  struct SegmentTree {
3      int sz;
4      T tr[N<<2], lazy[N<<2];
5      SegmentTree(){}
6      void build(const int &n, const T &k = 0) { sz = n; _build(1, n, k); }
7      template <typename TT>
8      void build(const TT a[], const int &n) { sz = n; _build(a, 1, n); }
9      void modify(const int &x, const T &k) { _modify(x, k, 1, sz); }
10     void add(const int &x, const T &k) { _add(x, x, k, 1, sz); }
11     void add(int l, int r, const T &k) { if (l > r) swap(l, r); _add(l, r, k, 1,
12         sz); }
12     T query(const int &x) { return _query(x, x, 1, sz); }
13     T query(int l, int r) { if (l > r) swap(l, r); return _query(l, r, 1, sz); }
14 private :
15     void push_up(const int &i) { tr[i] = tr[i<<1]+tr[i<<1|1]; }
16     void push_down(const int &i, const int &len) {
17         if (!lazy[i]) return;
18         tr[i<<1] += lazy[i]*(len-len/2);
19         tr[i<<1|1] += lazy[i]*(len/2);
20         lazy[i<<1] += lazy[i];
21         lazy[i<<1|1] += lazy[i];
22         lazy[i] = 0;
23     }
24     void _build(const int &l, const int &r, const T &k = 0, const int &i = 1) {
25         lazy[i] = 0;
26         if (l == r) { tr[i] = k; return; }
27         int mid = (l+r)>>1;
28         _build(l, mid, k, i<<1);
29         _build(mid+1, r, k, i<<1|1);
30         push_up(i);
31     }
32     template <typename TT>
33     void _build(const TT a[], const int &l, const int &r, const int &i = 1) {
34         lazy[i] = 0;
35         if (l == r) { tr[i] = a[l]; return; }
36         int mid = (l+r)>>1;
37         _build(a, l, mid, i<<1);
38         _build(a, mid+1, r, i<<1|1);
39         push_up(i);
40     }
41     void _modify(const int &x, const T &k, const int &trl, const int &trr, const
42         int &i = 1) {
43         if (trl == x && trr == x) {
44             tr[i] = k;
45             lazy[i] = 0;

```

```

45     return;
46 }
47 push_down(i, trr-trl+1);
48 int mid = (trl+trr)>>1;
49 if (x <= mid) _modify(x, k, trl, mid, i<<1);
50 else _modify(x, k, mid+1, trr, i<<1|1);
51 push_up(i);
52 }
53 void _add(const int &l, const int &r, const T &k, const int &trl, const int &
    trr, const int &i = 1) {
54     if (trl >= l && trr <= r) {
55         tr[i] += k*(trr-trl+1);
56         lazy[i] += k;
57         return;
58     }
59     push_down(i, trr-trl+1);
60     int mid = (trl+trr)>>1;
61     if (l <= mid) _add(l, r, k, trl, mid, i<<1);
62     if (r > mid) _add(l, r, k, mid+1, trr, i<<1|1);
63     push_up(i);
64 }
65 T _query(const int &l, const int &r, const int &trl, const int &trr, const
    int &i = 1) {
66     if (trl >= l && trr <= r) return tr[i];
67     push_down(i, trr-trl+1);
68     int mid = (trl+trr)>>1;
69     T res = 0;
70     if (l <= mid) res += _query(l, r, trl, mid, i<<1);
71     if (r > mid) res += _query(l, r, mid+1, trr, i<<1|1);
72     return res;
73 }
74 };

```

## 15.2 区间加减区间最值

```

1  template <typename T, typename U = greater<T>>
2  struct SegmentTree {
3      U cmp = U();
4      int n;
5      T tr[N<<2], lazy[N<<2], init_val = cmp(0, 1) ? INF : -INF;
6      SegmentTree(){}
7      T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
8      void build(const int &n, const T &k = 0) { n = _n; _build(1, n, k); }
9      template <typename TT>
10     void build(const TT a[], const int &n) { n = _n; _build(a, 1, n); }
11     void modify(const int &x, const T &k) { _modify(x, k, 1, n); }
12     void add(const int &x, const T &k) { _add(x, x, k, 1, n); }
13     void add(const int &l, const int &r, const T &k) { _add(l, r, k, 1, n); }
14     T query(const int &x) { return _query(x, x, 1, n); }
15     T query(const int &l, const int &r) { return _query(l, r, 1, n); }
16 private:
17     void push_up(const int &i) { tr[i] = mv(tr[i<<1], tr[i<<1|1]); }
18     void push_down(const int &i) {
19         if (!lazy[i]) return;
20         tr[i<<1] += lazy[i];
21         tr[i<<1|1] += lazy[i];
22         lazy[i<<1] += lazy[i];
23         lazy[i<<1|1] += lazy[i];
24         lazy[i] = 0;
25     }
26     void _build(const int &l, const int &r, const T &k = 0, const int &i = 1) {
27         lazy[i] = 0;
28         if (l == r) { tr[i] = k; return; }

```

```

29     int mid = (l+r)>>1;
30     _build(l, mid, k, i<<1);
31     _build(mid+1, r, k, i<<1|1);
32     push_up(i);
33 }
34 template <typename TT>
35 void _build(const TT a[], const int &l, const int &r, const int &i = 1) {
36     lazy[i] = 0;
37     if (l == r) { tr[i] = a[l]; return; }
38     int mid = (l+r)>>1;
39     _build(a, l, mid, i<<1);
40     _build(a, mid+1, r, i<<1|1);
41     push_up(i);
42 }
43 void _modify(const int &x, const T &k, const int &trl, const int &trr, const
44             int &i = 1) {
45     if (trl == x && trr == x) {
46         tr[i] = k;
47         return;
48     }
49     push_down(i);
50     int mid = (trl+trr)>>1;
51     if (x <= mid) _modify(x, k, trl, mid, i<<1);
52     else _modify(x, k, mid+1, trr, i<<1|1);
53     push_up(i);
54 }
55 void _add(const int &l, const int &r, const T &k, const int &trl, const int &
56          trr, const int &i = 1) {
57     if (trl >= l && trr <= r) {
58         tr[i] += k;
59         lazy[i] += k;
60         return;
61     }
62     push_down(i);
63     int mid = (trl+trr)>>1;
64     if (l <= mid) _add(l, r, k, trl, mid, i<<1);
65     if (r > mid) _add(l, r, k, mid+1, trr, i<<1|1);
66     push_up(i);
67 }
68 T _query(const int &l, const int &r, const int &trl, const int &trr, const
69          int &i = 1) {
70     if (trl >= l && trr <= r) return tr[i];
71     push_down(i);
72     int mid = (trl+trr)>>1;
73     T res = init_val;
74     if (l <= mid) res = mv(res, _query(l, r, trl, mid, i<<1));
75     if (r > mid) res = mv(res, _query(l, r, mid+1, trr, i<<1|1));
76     return res;
77 }
78 };

```

## 16 ZKW 线段树

warning: 区间最值尚为验证

```

1 template <typename T>
2 struct zkwSegmentTree {
3     int sz;
4     T sum[N<<2], mn[N<<2], mx[N<<2], add[N<<2];
5     void update(const int &x) {
6         T tmp;
7         tmp = min(mn[x], mn[x^1]); mn[x] -= tmp; mn[x^1] -= tmp; mn[x>>1] += tmp;
8         tmp = max(mx[x], mx[x^1]); mx[x] -= tmp; mx[x^1] -= tmp; mx[x>>1] += tmp;

```



```

9   }
10  template <typename TT>
11  void build(const TT a[], const int &n) {
12      for (sz = 1; sz <= n+1; sz <= 1);
13      for (int i = sz+1; i <= sz+n; ++i)
14          sum[i] = mn[i] = mx[i] = a[i-sz];
15      for (int i = sz-1; i; --i) {
16          sum[i] = sum[i<<1]+sum[i<<1|1];
17          mn[i] = min(mn[i<<1], mn[i<<1|1]); mn[i<<1] -= mn[i]; mn[i<<1|1] -= mn[i];
18          mx[i] = max(mx[i<<1], mx[i<<1|1]); mx[i<<1] -= mx[i]; mx[i<<1|1] -= mx[i];
19      }
20  }
21  void update(int x, const T &v) {
22      x += sz; mx[x] += v; mn[x] += v; sum[x] += v;
23      for (; x > 1; x >>= 1) {
24          sum[x] += v;
25          update(x);
26      }
27  }
28  void update(int s, int t, const T &v) {
29      int lc = 0, rc = 0, len = 1;
30      for (s += sz-1, t += sz+1; s^t^1; s >>= 1, t >>= 1, len <= 1) {
31          if (~s&1) add[s^1] += v, lc += len, mn[s^1] += v, mx[s^1] += v;
32          if (t&1) add[t^1] += v, rc += len, mn[t^1] += v, mx[t^1] += v;
33          sum[s>>1] += v*lc; sum[t>>1] += v*rc;
34          update(s); update(t);
35      }
36      for (lc += rc; s; s >>= 1) {
37          sum[s>>1] += v*lc;
38          update(s);
39      }
40  }
41  T query(int x) {
42      T res = 0;
43      for (x += sz; x; x >>= 1) res += mn[x];
44      return res;
45  }
46  T query_sum(int s, int t) {
47      int lc = 0, rc = 0, len = 1;
48      T res = 0;
49      for (s += sz-1, t += sz+1; s^t^1; s >>= 1, t >>= 1, len <= 1) {
50          if (~s&1) res += sum[s^1]+len*add[s^1], lc += len;
51          if (t&1) res += sum[t^1]+len*add[t^1], rc += len;
52          if (add[s>>1]) res += add[s>>1]*lc;
53          if (add[t>>1]) res += add[t>>1]*rc;
54      }
55      for (lc += rc, s >>= 1; s; s >>= 1) if (add[s]) res += add[s]*lc;
56      return res;
57  }
58  T query_min(int s, int t) {
59      if (s == t) return query(s);
60      T l = 0, r = 0, res = 0;
61      for (s += sz, t += sz; s^t^1; s >>= 1, t >>= 1) {
62          l += mn[s]; r += mn[t];
63          if (~s^1) l = min(l, mn[s^1]);
64          if (t^1) r = min(r, mn[t^1]);
65      }
66      for (res = min(l, r), s >>= 1; s; s >>= 1) res += mn[s];
67      return res;
68  }
69  T query_max(int s, int t) {
70      if (s == t) return query(s);
71      T l = 0, r = 0, res = 0;

```

```

72     for (s += sz, t += sz; s^t^1; s >>= 1, t >>= 1) {
73         l += mx[s]; r += mx[t];
74         if (~s^1) l = max(l, mx[s^1]);
75         if (t^1) r = max(r, mx[t^1]);
76     }
77     for (res = max(l, r), s >>= 1; s; s >>= 1) res += mx[s];
78     return res;
79 }
80 };

```

## 17 树状数组

### 17.1 一维

```

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() { for (int i = 1; i <= n; ++i) tr[i] = 0; }
8      void add(const int &x, const T &v) { for (int i = x; i <= n; i += i&-i) tr[i]
9          += v; }
10     void add(const int &x, const int &y, const T &v) { add(x, v); add(y+1, -v); }
11     T query(const int &x) { T res = 0; for (int i = x; i; i -= i&-i) res += tr[i]
12         ]; return res; }
13     T query(const int &x, const int &y) { return query(y)-query(x-1); }
14 };

```

$O(n)$  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  }

```

### 17.2 二维

#### 17.2.1 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 };

```

## 18 可持久化线段树 (主席树)

```

1  template <typename T>
2  struct PersistenceSegmentTree {
3      static const int NN = N*(log2(N)+5);
4      int rt[N], sum[NN], ls[NN], rs[NN], tot, sz;
5      vector<T> des;
6      void build(const T a[], const int &n) {
7          vector<T>(a+1, a+n+1).swap(des);
8          sort(des.begin(), des.end());
9          des.erase(unique(des.begin(), des.end()), des.end());
10         sz = des.size();
11         tot = 0;
12         rt[0] = _build(1, sz);
13         for (int i = 1; i <= n; ++i) {
14             int t = lower_bound(des.begin(), des.end(), a[i]) - des.begin() + 1;
15             rt[i] = _update(rt[i-1], 1, sz, t);
16         }
17     }
18     void update(const int &id, const T &k) {
19         int t = lower_bound(des.begin(), des.end(), k) - des.begin() + 1;
20         rt[id] = _update(rt[id-1], 1, sz, t);
21     }
22     T query(const int &l, const int &r, const int &k) {
23         return des[_query(rt[l-1], rt[r], 1, sz, k) - 1];
24     }
25 private:
26     int _build(const int &l, const int &r) {
27         int cur = ++tot;
28         sum[cur] = 0;
29         if (l >= r) return cur;
30         int mid = (l+r)>>1;
31         ls[cur] = _build(l, mid);
32         rs[cur] = _build(mid+1, r);
33         return cur;
34     }
35     int _update(const int &pre, const int &l, const int &r, const int &k) {
36         int cur = ++tot;
37         ls[cur] = ls[pre]; rs[cur] = rs[pre]; sum[cur] = sum[pre] + 1;
38         if (l >= r) return cur;
39         int mid = (l+r)>>1;
40         if (k <= mid) ls[cur] = _update(ls[pre], l, mid, k);
41         else rs[cur] = _update(rs[pre], mid+1, r, k);
42         return cur;
43     }

```

```

44 int _query(const int &u, const int &v, const int &l, const int &r, const int
    &k) {
45     if (l >= r) return l;
46     int num = sum[ls[v]]-sum[ls[u]], mid = (l+r)>>1;
47     if (num >= k) return _query(ls[u], ls[v], l, mid, k);
48     else return _query(rs[u], rs[v], mid+1, r, k-num);
49 }
50 };

```

## 19 分块

```

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

```

## 20 ST 表

### 20.1 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 };
29 /* rmq[i][j] ==> [i-2^j+1, i]
30 void build(T a[], const int &n) {
31     for (int i = 1; i <= n; ++i) {
32         rmq[i][0] = a[i];
33         for (int j = 1; j <= lg2[i]; ++j)
34             rmq[i][j] = mv(rmq[i][j-1], rmq[i-(1<<(j-1))][j-1]);
35     }
36 }
37 T query(const int &l, const int &r) {
38     if (l > r) return query(r, l);
39     int k = lg2[r-l+1];
40     return mv(rmq[r][k], rmq[l+(1<<k)-1][k]);
41 }
42 */

```

## 20.2 二维

```

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     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
11        : -INF); }
12    void init(const T &val) { fill(rmq[0][0][0], rmq[0][0][0]+N*N*NN*NN, val); }
13    T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
14    void build(T a[N][N], const int &n, const int &m) {
15        for (int k = 0; k <= log_2[n]; ++k)
16            for (int l = 0; l <= log_2[m]; ++l)
17                for (int i = 1; i+(1<<k)-1 <= n; ++i)
18                    for (int j = 1; j+(1<<l)-1 <= m; ++j) {
19                        T &cur = rmq[i][j][k][l];
20                        if (!k && !l) cur = a[i][j];
21                        else if (!l) cur = mv(rmq[i][j][k-1][l], rmq[i+(1<<(k-1))][j][k-1][l]);
22                        else cur = mv(rmq[i][j][k][l-1], rmq[i][j+(1<<(l-1))][k][l-1]);
23                    }
24    }
25    T query(const int &r1, const int &c1, const int &r2, const int &c2) {
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    }
30 };

```

## 20.3 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[r][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 };

```

## 21 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) {
7         int fx = get(x), fy = get(y);
8         if (fx == fy) return false;
9         fa[fx] = fy; return true;
10    }
11 } dsu;

```

加上按秩合并

```

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] = 1; }
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) {
7         int fx = get(x), fy = get(y);

```

```

8   if (fx == fy) return false;
9   if (num[fx] >= num[fy]) num[fx] += num[fy], fa[fy] = fx;
10  else num[fy] += num[fx], fa[fx] = fy;
11  return true;
12  }
13 } dsu;

```

## 第四部分 字符串

### 22 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 }

```

#### 22.1 判断 $s[l, r]$ 是否为回文

```

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

```

## 23 KMP

```

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

```

```

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 }

```

## 24 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), lensss = 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 < lensss && 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 }

```

## 25 字符串哈希

```

1 inline unsigned long long _hash(const string &s) {
2     unsigned long long res = 0;
3     for(int i = 0; i < s.length(); ++i)
4         res = (res*Base+s[i])%Mod+Prime;
5     return res;

```



6 }

## 26 后缀数组 |SA

### 26.1 $O(n\log^2 n)$

```

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, [&](const int &x, const int &y) {
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 }
```

### 26.2 $O(n\log n)$

```

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     #define cmp(x, y, w) oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w]
5     static int oldrk[N<<1], id[N], px[N], cnt[N], m;
6     memset(cnt, 0, sizeof(int) * (m = 128));
7     for (int i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
8     for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
9     for (int i = n; i; --i) sa[cnt[rk[i]]--] = i;
10    for (int w = 1, p, i; w <= n; w <= 1, m = p) {
11        for (p = 0, i = n; i > n - w; --i) id[++p] = i;
12        for (int i = 1; i <= n; ++i)
13            if (sa[i] > w)
14                id[++p] = sa[i] - w;
15        memset(cnt + 1, 0, sizeof(int) * m);
16        for (int i = 1; i <= n; ++i) ++cnt[px[i] = rk[id[i]]];
17        for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
18        for (int i = n; i; --i) sa[cnt[px[i]]--] = id[i];
19        memcpy(oldrk + 1, rk + 1, sizeof(int) * 2 * n);
20        for (p = 0, i = 1; i <= n; ++i) rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p :
21            ++p;
22    }
23    for (int i = 1, k = 0; i <= n; ++i) {
24        if (k) --k;
25    }
```

```

24     while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;
25     height[rk[i]] = k;
26 }
27 #undef cmp
28 }

```

### 26.3 $O(n)$

```

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,
10     bool *t, int *b, int *cb, int *p) {
11     memset(b, 0, sizeof(int) * sigma);
12     memset(sa, -1, sizeof(int) * len);
13     for (register int i = 0; i < len; i++) b[static_cast<int>(s[i])]++;
14     cb[0] = b[0];
15     for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i];
16     for (register int i = sz - 1; i >= 0; i--) sa[--cb[static_cast<int>(s[p[i]])]] = p[i];
17     for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i - 1];
18     for (register int i = 0; i < len; i++)
19         if (sa[i] > 0 && !t[sa[i] - 1])
20             sa[cb[static_cast<int>(s[sa[i] - 1])]++] = sa[i] - 1;
21     cb[0] = b[0];
22     for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i];
23     for (register int i = len - 1; i >= 0; i--)
24         if (sa[i] > 0 && t[sa[i] - 1])
25             sa[--cb[static_cast<int>(s[sa[i] - 1])]] = sa[i] - 1;
26 }
27
28 template <class T>
29 inline void sais(T s, int *sa, const int len, bool *t, int *b, int *b1, const
30     int sigma) {
31     register int i, j, x, p = -1, cnt = 0, sz = 0, *cb = b + sigma;
32     for (t[len - 1] = 1, i = len - 2; i >= 0; i--) t[i] = s[i] < s[i + 1] || (s[i]
33         ] == s[i + 1] && t[i + 1]);
34     for (i = 1; i < len; i++)
35         if (t[i] && !t[i - 1])
36             b1[sz++] = i;
37     sort(s, sa, len, sz, sigma, t, b, cb, b1);
38     for (i = sz = 0; i < len; i++)
39         if (islms(sa[i], t))
40             sa[sz++] = sa[i];
41     for (i = sz; i < len; i++) sa[i] = -1;
42     for (i = 0; i < sz; i++) {
43         for (x = sa[i], j = 0; j < len; j++) {
44             if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) {
45                 cnt++, p = x;
46                 break;
47             } else if (j > 0 && (islms(x + j, t) || islms(p + j, t))) {
48                 break;
49             }
50         }
51         sa[sz + (x >>= 1)] = cnt - 1;
52     }
53     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+1]] = 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

```

## 27 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 };

```

## 28 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;

```

## 第五部分 图论 | 树论

### 29 DFS 树

### 30 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 }

```

### 31 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;

```

### 32 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>(q).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            [&](const int &x, const int &y) { return wb[i][y] < wb[i][x]; });
16        //sort(rkg[i]+1, rkg[i]+n+1,
17        //    [&](const int &x, const int &y) { return wg[i][y] < wg[i][x]; });
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 };

```

## 33 最小生成树

## 34 二分图

### 34.1 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 }

```

### 34.2 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。

### 34.3 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

## 35 LCA

```

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 };

```

带权 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 };

```

## 36 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 };

```

## 37 树链剖分

```

1 template <typename T>
2 struct ShuPou {
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 &rt = 1) {
9         fa[rt] = dfn = 0;
10        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 };

```

## 38 网络流

### 38.1 最大流

#### 38.1.1 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 };

```

### 38.1.2 ISAP

渐进时间复杂度和 dinic 相同，但是非二分图的情况下 isap 更具优势

```

1 template <typename T>
2 struct ISAP {
3     struct EDGE

```

```

4  {
5      int v, nex;
6      T w;
7      EDGE(const int &_v, const int &_nex, const T &_w) : v(_v), nex(_nex), w(_w)
8          {}
9  };
10 vector<EDGE> e;
11 int n, s, t;
12 T maxflow;
13 int fir[N], gap[N], dep[N];
14 T work(const int &_s, const int &_t) {
15     s = _s; t = _t;
16     maxflow = 0;
17     bfs();
18     while (dep[s] < n) dfs(s, INF);
19     return maxflow;
20 }
21 void init(const int &n) {
22     n = _n;
23     e.clear();
24     e.reserve(N<<2);
25     memset(fir, -1, sizeof(int)*(n+3));
26 }
27 void add_edge(const int &u, const int &v, const T &w) {
28     e.emplace_back(v, fir[u], w); fir[u] = e.size()-1;
29     e.emplace_back(u, fir[v], 0); fir[v] = e.size()-1;
30 }
31 void bfs() {
32     queue<int> q;
33     memset(dep, -1, sizeof(int)*(n+3));
34     memset(gap, 0, sizeof(int)*(n+3));
35     dep[t] = 0;
36     gap[0] = 1;
37     q.push(t);
38     while (q.size()) {
39         int u = q.front();
40         q.pop();
41         for (int i = fir[u], v; i != -1; i = e[i].nex) {
42             v = e[i].v;
43             if (dep[v] != -1) continue;
44             q.push(v);
45             dep[v] = dep[u]+1;
46             ++gap[dep[v]];
47         }
48     }
49 }
50 T dfs(const int &u, const T &flow) {
51     if (u == t) {
52         maxflow += flow;
53         return flow;
54     }
55     T used = 0;
56     for (int i = fir[u], v; i != -1; i = e[i].nex) {
57         v = e[i].v;
58         if (!e[i].w || dep[v]+1 != dep[u]) continue;
59         T minf = dfs(v, min(e[i].w, flow-used));
60         if (minf) {
61             e[i].w -= minf;
62             e[i^1].w += minf;
63             used += minf;
64         }
65         if (used == flow) return used;
66     }
67     if (--gap[dep[u]] == 0) dep[s] = n+1;
68     ++gap[dep[u]];

```

```

68     return used;
69 }
70 };

```

### 38.1.3 HLPP

## 38.2 最小割

最小割等价最大流

## 38.3 费用流

### 38.3.1 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 };

```

## 38.4 上下界网络流

## 39 最短路

### 39.1 Floyd

### 39.2 Dijiskra

### 39.3 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 }

```

## 40 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 }

```

## 41 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 }

```

## 42 SCC 强连通分量 | Tarjan

### 42.1 递归版本

```

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 }

```

## 43 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 }

```

## 44 2-SAT

### 44.1 SCC Tarjan

$O(n + m)$  从 0 开始

```

1 struct TWO_SAT {
2   int top, _dfn, _col;
3   int dfn[N<<1], low[N<<1], vis[N<<1], sta[N<<1], col[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(vis, 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 &vv) {
17     e[u<<1^uv].emplace_back(v<<1^vv);
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 (col[i<<1] == col[i<<1|1]) return false;
25         res[i] = col[i<<1] > col[i<<1|1];
26     }
27     return true;
28 }
29 void tarjan(const int &u) {
30     dfn[u] = low[u] = ++_dfn;
31     vis[u] = 1;
32     sta[++top] = u;
33     for (int &v : e[u]) {
34         if (!dfn[v]) {
35             tarjan(v);
36             low[u] = min(low[u], low[v]);
37         } else if (vis[v]) {
38             low[u] = min(low[u], low[v]);
39         }
40     }
41     if (dfn[u] == low[u]) {
42         ++_col;
43         do {
44             col[sta[top]] = _col;
45             vis[sta[top]] = 0;
46         } while (sta[top--] != u);
47     }
48 }
49 };

```

## 44.2 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 };

```

## 第六部分 数论

### 45 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 }

```

### 46 求第 K 大数

```

1 int kth_element(int l, int r, int k) {
2     if(l == r) return a[l];
3     swap(a[l], a[l+rand()%(r-l)]);
4     int mid = a[l], i = l, j = r;
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    a[i] = mid;
12    if(i == k) return mid;
13    else if(i > k) return kth_element(l, i-1, k);
14    else return kth_element(i+1, r, k);
15 }

```

**\*\*STL\*\*** (排序, 无返回值)

```

1 nth_element(a+1, a+k+1, a+n+1);

```

## 47 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 = l;
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+1, b+1, sizeof(int)*(r-l+1));
15 }

```

## 48 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 };

```

```

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 };

```

## 49 矩阵

### 49.1 矩阵快速幂

### 49.2 矩阵求逆

```

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)%MOD;
46             for (int i = 1; i <= n; ++i) if (i != k)
47                 res.a[i][k] = (MOD-res.a[i][k]*res.a[k][k]%MOD)%MOD;
48         }
49         for (int k = n; k; --k) {
50             for (int i = 1; i <= n; ++i) swap(res.a[js[k]][i], res.a[k][i]);
51             for (int i = 1; i <= n; ++i) swap(res.a[i][is[k]], res.a[i][k]);
52         }
53         return res;
54     }
55     T det() {
56         long long res = 1;
57         Martix cpy = *this;
58         for (int i = 1; i <= n; ++i) {
59             for (int j = i+1; j <= n; ++j) while (cpy.a[j][i]) {

```

```

60     long long t = cpy.a[i][i]/cpy.a[j][i];
61     for (int k = i; k <= n; ++k)
62         cpy.a[i][k] = (cpy.a[i][k]+MOD-t*cpy.a[j][k]%MOD)%MOD;
63     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 };

```

## 50 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 };

```

### 50.1 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 }

```

## 51 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      long long res = 0, s1 = 1, s2 = 1;
4      for (int i = 1; i <= n; ++i, s1 = s2 = 1) {
5          for (int j = 1; j <= n; ++j) if (i != j) {
6              s1 = s1*(x[i]-x[j]+MOD)%MOD;
7              s2 = s2*(k-x[j]+MOD)%MOD;
8          }
9          res = (res+y[i]*s2%MOD*mul_inverse(s1)%MOD)%MOD;
10     }
11     return res;
12 }

```

```

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      pre[0] = suf[n+1] = 1;
7      for (int i = 1; i <= n; ++i) pre[i] = pre[i-1]*(k-i)%MOD;
8      for (int i = n; i >= 1; --i) suf[i] = suf[i+1]*(k-i)%MOD;
9      for (int i = 1; i <= n; ++i) {
10         res = (res+y[i]*(pre[i-1]*suf[i+1]%MOD)%MOD
11             *mul_inverse(((n-i)&1 ? -1 : 1)*fac[i-1]*fac[n-i]%MOD)%MOD)%MOD;
12     }
13     return (res+MOD)%MOD;
14 }

```

## 52 快速幂

```

1  template <typename T, typename H>
2  inline T qpow(const T &a, const H &p, const int &mo = MOD) {
3      long long res = 1, x = a;
4      for (H i = p; i; i >>= 1, x = x*x%mo)
5          if (i&1) res = res*x%mo;
6      return static_cast<T>(res);
7  }

```

## 53 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 }

```

## 54 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(_real), imag(
5         _imag) {}
6     friend comp operator + (const comp &c1, const comp &c2) { return comp(c1.real
7         +c2.real, c1.imag+c2.imag); }
8     friend comp operator - (const comp &c1, const comp &c2) { return comp(c1.real
9         -c2.real, c1.imag-c2.imag); }
10    friend comp operator * (const comp &c1, const comp &c2) { return comp(c1.real
11        *c2.real-c1.imag*c2.imag, c1.real*c2.imag+c1.imag*c2.real); }
12    comp& operator += (const comp &c) { return *this = *this+c; }
13    comp& operator -= (const comp &c) { return *this = *this-c; }
14    comp& operator *= (const comp &c) { return *this = *this*c; }
15    friend istream& operator >> (istream &is, comp &c) { return is >> c.real >> c
16        .imag; }
17    friend ostream& operator << (ostream &os, comp &c) { return os << c.real <<
18        setiosflags(ios::showpos) << c.imag << "i"; }
19    comp conjugate() { return comp(real, -imag); }
20    friend comp conjugate(const comp &c) { return comp(c.real, -c.imag); }
21 };

```

## 55 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 /*
37 template <class T>
38 void work(T a[], const int &n) {
39     static comp f[SIZE];
40     len = 1; bit = 0;
41     while (len < n+n) len <= 1, ++bit;
42     // multi-testcase
43     for (int i = 0; i < n; ++i) f[i] = a[i];
44     for (int i = n; i < len; ++i) f[i] = 0;
45     for (int i = 0; i < len; ++i) rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
46     fft(f, 1);
47     for (int i = 0; i <= len; ++i) f[i] *= f[i];
48     fft(f, -1);
49     for (int i = 0; i < n+n; ++i) a[i] = static_cast<T>(f[i].real/len+.5);
50 }
51 */
52 }

```

## 56 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 }

```

## 57 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[bitrev[i]]);
9         for (int i = 2, lyc = n >> 1; i <= n; i <= 1, lyc >>= 1)
10             for (int j = 0; j < n; j += i) {
11                 comp *l = a + j, *r = a + j + (i >> 1), *p = w;
12                 for (int k = 0; k < i>>1; ++k) {
13                     comp tmp = *r * *p;
14                     *r = *l - tmp, *l = *l + tmp;
15                     ++l, ++r, p += lyc;
16                 }
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];
24     static comp dfta[SIZE], dftb[SIZE];
25
26     for (L = 1, bit = 0; L < n+m-1; ++bit, L <= 1);
27     for (int i = 0; i < L; ++i) bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (bit - 1));
28     for (int i = 0; i < L; ++i) w[i] = comp(cos(2 * PI * i / L), sin(2 * PI * i / L));
29
30     for (int i = 0; i < n; ++i) (x[i] += Mod) %= Mod, a[i] = comp(x[i] & 32767, x[i] >> 15);
31     for (int i = n; i < L; ++i) a[i] = 0;
32     for (int i = 0; i < m; ++i) (y[i] += Mod) %= Mod, b[i] = comp(y[i] & 32767, y[i] >> 15);

```

```

32 for (int i = m; i < L; ++i) b[i] = 0;
33 fft(a, L), fft(b, L);
34 for (int i = 0; i < L; ++i) {
35     int j = (L - i) & (L - 1);
36     static comp da, db, dc, dd;
37     da = (a[i] + conjugate(a[j])) * comp(.5, 0);
38     db = (a[i] - conjugate(a[j])) * comp(0, -.5);
39     dc = (b[i] + conjugate(b[j])) * comp(.5, 0);
40     dd = (b[i] - conjugate(b[j])) * comp(0, -.5);
41     dfta[j] = da*dc + da*dd*comp(0, 1);
42     dftb[j] = db*dc + db*dd*comp(0, 1);
43 }
44 for (int i = 0; i < L; ++i) a[i] = dfta[i];
45 for (int i = 0; i < L; ++i) b[i] = dftb[i];
46 fft(a, L), fft(b, L);
47 for (int i = 0; i < L; ++i) {
48     int da = (long long)(a[i].real / L + 0.5) % Mod;
49     int db = (long long)(a[i].imag / L + 0.5) % Mod;
50     int dc = (long long)(b[i].real / L + 0.5) % Mod;
51     int dd = (long long)(b[i].imag / L + 0.5) % Mod;
52     f[i] = (da + ((long long)(db + dc) << 15) + ((long long)dd << 30)) % Mod;
53 }
54 for (int i = 0; i < n+m-1; ++i) (f[i] += Mod) %= Mod;
55 }
56 }

```

## 58 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);

```

## 59 第二类斯特林数

```

1 inline void stirling(const int &n) {
2     S[0][0] = 1;
3     // 注意取模
4     for (int i = 1; i <= n; ++i)
5         for (int j = 1; j <= i; ++j)
6             S[i][j] = S[i-1][j-1] + S[i-1][j]*j;
7 }

```

```

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 }

```

## 60 约瑟夫环

### 60.1 $O(n)$

```

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

```

## 61 最大公因数 gcd

```

1  __gcd(a, b); // <algorithm>
2  int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
3  inline int gcd(int a, int b) { while (b) a %= b, swap(a, b); return a; }

```

## 62 最小公倍数 lcm

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

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

```

1  inline int lcm(int a, int b) { return a/gcd(a, b)*b; }

```

## 63 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) { x = 1; y = 0; return a; }
4      T d = exgcd(b, a%b, y, x);
5      y -= a/b*x;
6      return d;
7  }

```

## 64 乘法逆元

### 64.1 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 }

```

## 64.2 费马小定理

```

1 template <typename T>
2 inline T mul_inverse(const T &a, const int &mo = MOD) {
3     return qpow(a, mo-2);
4 }

```

## 64.3 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 }

```

# 65 中国剩余定理

## 65.1 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 }

```

## 65.2 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 }

```

## 66 排列组合

### 66.1 奇偶性

$C(n, k)$  当  $n \& k == k$  为奇数反之偶数

## 67 欧拉函数

```

1 inline long long phi(long long x) {
2     long long res = x;
3     for (long long 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 }

```

### 67.1 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;

```

## 68 线性筛

```

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;

```

## 69 判断素数（质数）

某较优方法

```

1 inline bool is_prime(long long x) {
2     if(x == 1) return false;
3     if(x == 2 || x == 3) return true;
4     if(x%6 != 1 && x%6 != 5) return false;
5     for(long long i = 5; i*i <= x; i += 6)
6         if(x%i == 0 || x%(i+2) == 0) return false;
7     return true;
8 }

```

### 69.1 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 }

```

## 70 BSGS

```

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     {
10         now = qpow(a, i);
11         if(mmp.count(now))
12         {
13             ans = t*i-mmp[now];
14             if(ans > 0) return ans;
15         }
16     }
17     return -1;
18 }

```

## 第七部分 动态规划 DP

### 71 线性 DP

#### 71.1 最长上升子序列 LIS

```

1 for(int i = 1; i <= n; ++i) {
2     f[i] = 1;
3     for(int j = 1; j < i; ++j)
4         if(a[i] > a[j]) f[i] = max(f[i], f[j]+1);
5 }

```

#### 71.2 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])}

```

#### 71.3 数字三角形

### 72 状压 DP

#### 72.1 枚举子集

```

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

```

#### 72.2 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 }

```

### 73 背包问题

#### 73.1 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)
21     ans = max(ans, f[i]);

```

## 第八部分 STL

### 74 unordered\_map 重载

```

1 struct Node {
2     int a, b;
3     // 重载 ==
4     friend bool operator == (const Node &x, const Node &y) {
5         return x.a == y.a && x.b == y.b;
6     }
7 };
8 // 方法一
9 namespace std {
10     template <>
11     struct hash<Node> {
12         size_t operator () (const Node &x) const {
13             return hash<int>()(x.a)^hash<int>()(x.b);
14         }
15     };
16 }
17 unordered_map<Node, int> mp;
18 // 方法二
19 struct KeyHasher {
20     size_t operator () (const Node &x) const {
21         return hash<int>()(x.a)^hash<int>()(x.b);
22     }
23 };
24 unordered_map<Node, int, KeyHasher> mmp;

```

### 75 定义函数

```

1 function<void(int&, int)> f = [&](int &x, int y) -> void {
2     x += y;
3 };

```