目录

Part I
杂项

1 快读快写

2 正则表达式

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

3 随机数

```
#include <random>
// 范围 unsigned int
mt19937 rnd(time(NULL));
mt19937 rnd(chrono::high_resolution_clock::now().time_since_epoch().
   count());

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

inline ull xorshift128(){
  static U SX=335634763,SY=873658265,SZ=192849106,SW=746126501;
  U t=SX^(SX<<11);
  SX=SY;
  SY=SZ;
  SZ=SW;
  return SW=SW^(SW>>19)^t^(t>>8);
}
inline ull myrand(){return (xorshift128()<<32)^xorshift128();}
```

4 计算 log2

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

5 快速开根号 | 牛顿迭代法

```
double sqrt(const double &a) {
  double x = a, y = .0;
  while (abs(x-y) > err) {
    y = x;
    x = .5*(x+a/x);
  }
  return x;
}
```

6 i/k == j 的 k 的个数

```
for (int i = 1; i <= n; ++i) {
  for (int j = 1, l, r; j <= n; ++j) {
    l = max(1, i/(j+1));
    while (l-1 >= 1 && i/(l-1) == j) --l;
    while (i/l > j) ++l;
    r = i/j;
    while (r+1 <= i && i/(r+1) == j) ++r;
    while (i/r < j) --r;
    if (r-l+1 != i/j-i/(j+1)) {
        cout << i << " " << j << endl;
    }
  }
}
```

7 三分法

示例为凹函数

```
while (l < r) {
  int mid = (l+r)>>1;
  if (f(mid) < f(mid+1)) r = mid;
  else l = mid+1;
}
```

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

```
while (r-l > eps) {
  double ml = l+(r-l)/3, mr = r-(r-l)/3;
  if (f(ml) < f(mr)) r = mr;
  else l = ml;
}
```

8 三维偏序 |CDQ 分治

```
void cdq(int l, int r) {
  if (l >= r) return;
  int mid = (l+r)>>1, i = l, j = mid+1;
  cdq(l, mid); cdq(mid+1, r);
  for (int k = l; k <= r; ++k) {
    if (j > r || (i <= mid && a[i].y <= a[j].y)) {
      tree.add(a[i].z, a[i].w);
      b[k] = a[i++];
    } else {
      a[j].f += tree.query(a[j].z);
      b[k] = a[j++];
    }
  }
  for (int k = l; k <= mid; ++k) tree.add(a[k].z, -a[k].w);
  for (int k = l; k <= r; ++k) a[k] = b[k];
}
```

8.1 四维偏序

CDQ 套 CDQ
第一维在第一层 CDQ 合并时标记左右区间
合并后第二维有序，进入第二层 CDQ
此时按照正常 CDQ 合并第三维，用数据结构统计第四维
只有标记左区间的加入数据结构，标记右边区间的更新答案

```
void cdq(int l, int r) {
  cdq(l, mid); cdq(mid+1, r);
}

void CDQ(int l, int r) {
  CDQ(l, mid); CDQ(mid+1, r);
  // ...
  cdq(l, r);
}
```

9 某区间操作问题

每次操作可以使一个区间 +1(或-1)，使得序列为 0 的操作次数下界 $\frac{\sum_{i=1}^{n+1}|a_i-a_{i-1}|}{2}$

10 分数规划

分数规划用来求一个分式的极值。
形象一点就是，给出 a_i 和 b_i，求一组 $w_i \in 0,1$，最小化或最大化

$$\frac{\sum_{i=1}^{n} a_i \times w_i}{\sum_{i=1}^{n} b_i \times w_i}$$

每种物品有两个权值 a 和 b，选出若干个物品使得 $\frac{\sum a}{\sum b}$ 最小/最大。

例如代价为平均值或乘积开个数次根（取 ln）

分数规划问题的通用方法是二分。

假设我们要求最大值。二分一个答案 mid，然后推式子（为了方便少写了上下界）：

$$\begin{aligned} &\frac{\sum a_i \times w_i}{\sum b_i \times w_i} > mid \\ \Longrightarrow &\sum a_i \times w_i - mid \times \sum b_i \cdot w_i > 0 \\ \Longrightarrow &\sum w_i \times (a_i - mid \times b_i) > 0 \end{aligned}$$

那么只要求出不等号左边的式子的最大值就行了。如果最大值比 0 要大，说明 mid 是可行的，否则不可行。

Part II
计算几何

11 向量坐标直线圆（结构体）

```
struct Point {
  typedef double T;
  T x, y;
  int id;
  Point(){}
  Point(const T &_x, const T &_y, const int &_i = 0) : x(_x), y(_y),
     id(_i) {}
  friend Point operator + (const Point &p1, const Point &p2) {
    return Point(p1.x+p2.x, p1.y+p2.y, p1.id);
  }
  friend Point operator - (const Point &p1, const Point &p2) {
    return Point(p1.x-p2.x, p1.y-p2.y, p1.id);
  }
  friend Point operator - (const Point &p) {
    return Point(-p.x, -p.y, p.id);
  }
  // a*b b在a的顺负逆正
  friend T operator * (const Point &p1, const Point &p2) {
    return p1.x*p2.y-p1.y*p2.x;
  }
  template <typename TT>
  friend Point operator / (const Point &p, const TT &k) {
    return Point(p.x/k, p.y/k, p.id);
  }
  template <typename TT>
  friend Point operator * (const Point &p, const TT &k) {
    return Point(p.x*k, p.y*k, p.id);
  }
  Point operator += (const Point &p) { return *this = *this+p; }
  Point operator -= (const Point &p) { return *this = *this+p; }
  template <typename TT>
  Point operator *= (const TT &k) { return *this = *this*k; }
```

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

```
  Vec& operator -=(const Vec &v) { return *this = *this-v; }
  Vec& operator *=(const T &k) { return *this = *this*k; }
  Vec& operator *=(const Vec &v) { return *this = *this*v; }
  friend istream& operator >>(istream &is, Vec &v) { return is >> v.x >> v.y >> v.z; }
};

inline bool polar_angle1(const Point &p1, const Point &p2) {
  double d1 = atan(p1), d2 = atan(p2);
  return d1 == d2 ? p1 < p2 : d1 < d2;
}

inline bool polar_angle2(const Point &p1, const Point &p2) {
  auto tmp = p1*p2;
  return tmp == 0 ? p1 < p2 : tmp > 0;
}

inline long long S(const Point &p1, const Point &p2, const Point &p3) {
  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);
}

struct Line {
  Point p1, p2;
  Line(){}
  Line(const Point &_p1, const Point &_p2) : p1(_p1), p2(_p2) {}
  friend bool cross(const Line &l1, const Line &l2) {
    #define SJ1(x) max(l1.p1.x, l1.p2.x) < min(l2.p1.x, l2.p2.x) || \
                   max(l2.p1.x, l2.p2.x) < min(l1.p1.x, l1.p2.x)
    if (SJ1(x) || SJ1(y)) return false;
    #undef SJ1
    #define SJ2(a, b, c, d) ((a-b)*(a-c))*((a-b)*(a-d)) <= 0
    return SJ2(l1.p1, l1.p2, l2.p1, l2.p2) &&
           SJ2(l2.p1, l2.p2, l1.p1, l1.p2);
    #undef SJ2
  }
  friend bool on_line(const Line &l, const Point &p) {
    return abs((l.p1-l.p2)*(l.p1-p)) < eps;
  }
  friend Point cross_point(const Line &l1, const Line &l2) {
    Point v1 = l1.p2-l1.p1, v2 = l2.p2-l2.p1;
    if (abs(v1*v2) < eps) return Point(0, 0); // no cross_point
    double t = (l2.p1-l1.p1)*v2/(v1*v2);
    return l1.p1+v1*t;
  }
};

struct Circular {
  Point o;
  double r;
  Circular(){}
  Circular(const Point &_o, const double &_r) : o(_o), r(_r) {}
  template <typename T>
  Circular(const T &_x, const T &_y, const double &_r) : o(Point(_x, _y)), r(_r) {}
  friend bool in_cir(const Circular &c, const Point &p) { return dis(
```

```
    c.o, p) <= c.r; }
  bool in_cir(const Point &p) { return dis(o, p) <= r; }
};

inline Circular get_cir(const Point &p1, const Point &p2, const Point &p3) {
  Circular res;
  res.o = cross_point(Line((p1+p2)/2, (p1+p2)/2+(p2-p1).rotate_90_c()),
            Line((p1+p3)/2, (p1+p3)/2+(p3-p1).rotate_90_c()));
  res.r = dis(res.o, p1);
  return res;
}
```

12 二维凸包

```
int n;
int stk[N], used[N], tp;
Point p[N];

inline void Andrew() {
  memset(used, 0, sizeof used);
  sort(p+1, p+n+1);
  tp = 0;
  stk[++tp] = 1;
  for (int i = 2; i <= n; ++i) {
    while (tp >= 2 && (p[stk[tp]]-p[stk[tp-1]])*(p[i]-p[stk[tp]]) <= 0)
      used[stk[tp--]] = 0;
    used[i] = 1;
    stk[++tp] = i;
  }
  int tmp = tp;
  for (int i = n-1; i; --i) {
    if (used[i]) continue;
    while (tp >= tmp && (p[stk[tp]]-p[stk[tp-1]])*(p[i]-p[stk[tp]]) <= 0)
      used[stk[tp--]] = 0;
    used[i] = 1;
    stk[++tp] = i;
  }
}
```

13 平面最近点对

```
Point a[N];
int n, ansa, ansb;
double mindist;

inline bool cmp_y(const Point &p1, const Point &p2) { return p1.y < p2.y; }

```

```
void upd_ans(const Point &p1, const Point &p2) {
  double dist = dis(p1, p2);
  if (dist < mindist) mindist = dist, ansa = p1.id, ansb = p2.id;
}

void rec(int l, int r) {
  if (r-l <= 3) {
    for (int i = l; i < r; ++i)
      for (int j = i+1; j <= r; ++j)
        upd_ans(a[i], a[j]);
    sort(a+l, a+r+1, cmp_y);
    return;
  }

  static Point t[N];
  int m = (l+r)>>1, midx = a[m].x;
  rec(l, m); rec(m+1, r);
  merge(a+l, a+m+1, a+m+1, a+r+1, t, cmp_y);
  copy(t, t+r-l+1, a+l);

  int tsz = 0;
  for (int i = l; i <= r; ++i)
    if (abs(a[i].x-midx) <= mindist) {
      for (int j = tsz; j && a[i].y-t[j].y < mindist; --j)
        upd_ans(a[i], t[j]);
      t[++tsz] = a[i];
    }
}

inline void mindist_pair() {
  sort(a+1, a+n+1);
  mindist = INF;
  rec(1, n);
}
```

14 最小圓覆盖 | 随即增量法

```
inline Circular RIA() {
  Circular cir;
  random_shuffle(a+1, a+n+1);
  for (int i = 1; i <= n; ++i) {
    if (cir.in_cir(a[i])) continue;
    cir = Circular(a[i], 0);
    for (int j = 1; j < i; ++j) {
      if (cir.in_cir(a[j])) continue;
      cir = Circular((a[i]+a[j])/2, dis(a[i], a[j])/2);
      for (int k = 1; k < j; ++k) {
        if (cir.in_cir(a[k])) continue;
        cir = get_cir(a[i], a[j], a[k]);
      }
    }
  }
  return cir;
}
```

Part III
数据结构

15 堆

```cpp
struct Heap {
  static const int Maxn = 1e6+7;
  int sz, a[Maxn];
  Heap() { sz = 0; memset(a, 0, sizeof a); }
  inline bool cmp(int x, int y) { return x < y; } // 小根堆
  inline int size() { return sz; }
  inline bool empty() { return sz == 0; }
  inline int top() { return a[1]; }
  inline void push(int x) { a[++sz] = x; swift_up(sz); }
  inline void pop() { swap(a[1], a[sz--]); swift_down(1); }
  inline void swift_up(int p) {
    while(p > 1 && cmp(a[p], a[p>>1])) // a[p] < a[p<<1]
      swap(a[p], a[p>>1]), p >>= 1;
  }
  inline void swift_down(int p) {
    int l, r, s;
    while(true) {
      l = p<<1; r = p<<1|1;
      if(l > sz) break;
      if(r > sz || cmp(a[l], a[r])) s = l; // a[l] < a[r]
      else s = r;
      if(cmp(a[s], a[p])) // a[s] < a[p]
        swap(a[p], a[s]), p = s;
      else break;
    }
  }
};
```

15.1 可删堆

copyright by axiomofchoice

```cpp
template <typename T>
struct Heap{
  priority_queue<T> a,b;  // heap=a-b
  void push(T x){a.push(x);}
  void erase(T x){b.push(x);}
  T top(){
    while(!b.empty() && a.top()==b.top())
      a.pop(),b.pop();
    return a.size() ? a.top() : 0; // ???
  }
  void pop(){
    while(!b.empty() && a.top()==b.top())
      a.pop(),b.pop();
    a.pop();
  }
  T top2(){ // 次大值
```

```
    T t=top(); pop();
    T ans=top(); push(t);
    return ans;
  }
  size_t size(){return a.size()-b.size();}
};
```

16 二叉查找树

17 平衡树

17.1 Splay

```
struct Splay {
  #define root e[0].ch[1]
  typedef int T;
  struct node {
    T v = 0;
    int ch[2] = {0, 0};
    int fa = 0, sum = 0, cnt = 0;
  } e[N];
  int n;
  void update(int x) { e[x].sum = e[e[x].ch[0]].sum+e[e[x].ch[1]].sum
     +e[x].cnt; }
  int identify(int x) { return x == e[e[x].fa].ch[1]; } // check left
      or right child
  void connect(int x, int f, int son) { e[x].fa = f; e[f].ch[son] = x
     ; }
  void rotate(int x) {
    int y = e[x].fa,
        r = e[y].fa,
        rson = identify(y),
        yson = identify(x),
        b = e[x].ch[yson^1];
    connect(b, y, yson);
    connect(y, x, yson^1);
    connect(x, r, rson);
    update(y); update(x);
  }
  void splay(int at, int to) {
    to = e[to].fa;
    int up;
    while((up = e[at].fa) != to) {
      if(e[up].fa != to)
        rotate(identify(up) == identify(at) ? up : at);
      rotate(at);
    }
  }
  int add_point(T v, int fa) {
    ++n; e[n].v = v; e[n].fa = fa; e[n].sum = e[n].cnt = 1;
    return n;
  }
  int find(T v) {
    int now = root, last = 0;
```

```
    while (now && e[now].v != v)
      last = now, now = e[now].ch[v > e[now].v];
    splay((now ? now : last), root);
    return now;
  }
  void insert(T v) {
    if (!root) { root = add_point(v, root); return; }
    int now = root, last = 0;
    while (now && e[now].v != v)
      last = now, now = e[now].ch[v > e[now].v];
    if (now) ++e[now].cnt;
    else now = e[last].ch[v > e[last].v] = add_point(v, last);
    splay(now, root);
  }
  void erase(T v) {
    int del = find(v);
    if (!del) return;
    if (e[del].cnt > 1) {
      --e[del].cnt;
      --e[del].sum;
    } else if (!e[del].ch[0]) {
      root = e[del].ch[1];
      e[root].fa = 0;
    } else {
      int oldroot = root;
      splay(nex(e[del].ch[0], 1), root);
      connect(e[oldroot].ch[1], root, 1);
      update(root);
    }
  }
  int rank(T v) { return e[e[find(v)].ch[0]].sum+1; }
  T atrank(int x) {
    if (x > e[root].sum) return -INF;
    int now = root;
    while (true) {
      if (x <= e[e[now].ch[0]].sum) now = e[now].ch[0];
      else if ((x -= e[e[now].ch[0]].sum) <= e[now].cnt) break;
      else x -= e[now].cnt, now = e[now].ch[1];
    }
    splay(now, root);
    return e[now].v;
  }
  // small 0, big 1
  int nex(int x, int opt) { while (e[x].ch[opt]) x = e[x].ch[opt];
     return x; }
  T lower(T v, int opt) {
    insert(v);
    T res = e[nex(e[root].ch[opt], opt^1)].v;
    erase(v);
    return res;
  }
  #undef root
};
```

区间反转

```
struct Splay {
  typedef int T;
```

```
3   struct node {
4     T v = 0;
5     int ch[2] = { 0, 0 };
6     int fa = 0, sum = 0, cnt = 0, tag = 0;
7   } e[N];
8   int sz, &root = e[0].ch[1];
9   void update(int x) { e[x].sum = e[e[x].ch[0]].sum+e[e[x].ch[1]].sum
      +e[x].cnt; }
10  int identify(int x) { return x == e[e[x].fa].ch[1]; }
11  void connect(int x,int f,int son) { e[x].fa = f; e[f].ch[son] = x;
      }
12  void rotate(int x) {
13    int y = e[x].fa,
14      r = e[y].fa,
15      rson = identify(y),
16      yson = identify(x),
17      b = e[x].ch[yson^1];
18    connect(b, y, yson);
19    connect(y, x, yson^1);
20    connect(x, r, rson);
21    update(y); update(x);
22  }
23  void splay(int at,int to = 0) {
24    to = e[to].fa;
25    int up;
26    while((up = e[at].fa) != to) {
27      if(e[up].fa != to)
28        rotate(identify(up) == identify(at) ? up : at);
29      rotate(at);
30    }
31  }
32  int add_point(T v, int fa) {
33    ++sz; e[sz].v = v; e[sz].fa = fa; e[sz].sum = e[sz].cnt = 1;
34    return sz;
35  }
36  int find(int x) {
37    if (x > e[root].sum) return -INF;
38    int now = root;
39    while (true) {
40      push_down(now);
41      if (x <= e[e[now].ch[0]].sum) now = e[now].ch[0];
42      else if ((x -= e[e[now].ch[0]].sum) <= e[now].cnt) break;
43      else x -= e[now].cnt, now = e[now].ch[1];
44    }
45    return now;
46  }
47  int build(int l, int r, int fa) {
48    if (l > r) return 0;
49    int mid = (l+r)>>1,
50      now = add_point(mid, fa);
51    e[now].ch[0] = build(l, mid-1, now);
52    e[now].ch[1] = build(mid+1, r, now);
53    update(now);
54    return now;
55  }
56  void push_down(int x) {
57    if (x && e[x].tag) {
```

```
      e[e[x].ch[0]].tag ^= 1;
      e[e[x].ch[1]].tag ^= 1;
      swap(e[x].ch[0], e[x].ch[1]);
      e[x].tag = 0;
    }
  }
  void reverse(int l, int r) {
    int pl = find(l-1+1), pr = find(r+1+1);
    splay(pl); splay(pr, pl);
    e[e[e[root].ch[1]].ch[0]].tag ^= 1;
  }
  void print_LMR(int x) {
    if (!x) return;
    push_down(x);
    print_LMR(e[x].ch[0]);
    if (e[x].v != 0 && e[x].v != n+1)
      write(a[e[x].v]), putchar(' ');
    print_LMR(e[x].ch[1]);
  }
} tree;
```

18 李超线段树

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

19 线段树合并

有 $O(m)$ 棵树（个操作）结点数量 $O(m \log n)$，暴力合并，均摊复杂度 $O(\log n)$

```
int merge(int x, int y) { // 太难了,现场全重写吧
  if (!x || !y) return x+y;
  int &z = x; // int z = new_node(); // 新建结点?
  lc[z] = merge(lc[x], lc[y]);
  rc[z] = merge(rc[x], rc[y]);
  if (!lc[z] && !rc[z]) tr[z].first = tr[x].first+tr[y].first;
  else push_up(z); // tr[z] = giao(x, y);
  // del(y); // del(x); // 保留结点?
  return z;
}
```

20 线段树分裂

一次分裂复杂度 $O(\log n)$

```
template <typename T>
struct Tree {
  int tot, lc[NLOG], rc[NLOG];
  T tr[NLOG];
  void init() { tot = 0; }
  T giao(const T &x, const T &y) { return x+y; }
```

```
7   void push_up(int i) { tr[i] = giao(tr[lc[i]], tr[rc[i]]); }
8   int new_node(T v = 0) {
9     ++tot; // assert(++tot < NLOG);
10    lc[tot] = rc[tot] = 0;
11    tr[tot] = v;
12    return tot;
13  }
14  void add(int x, T v, int l, int r, int &i) {
15    if (!i) i = new_node();
16    if (l == r) return tr[i] += v, void();
17    int mid = (l+r)>>1;
18    if (x <= mid) add(x, v, l, mid, lc[i]);
19    else add(x, v, mid+1, r, rc[i]);
20    push_up(i);
21  }
22  void merge(int l, int r, int &x, int &y) { // merge y to x
23    if (!x || !y) return x += y, void();
24    if (l == r) return tr[x] += tr[y], void();
25    int mid = (l+r)>>1;
26    merge(l, mid, lc[x], lc[y]);
27    merge(mid+1, r, rc[x], rc[y]);
28    push_up(x); // del(y);
29  }
30  /*
31  int merge(int l, int r, int x, int y) { // new node
32    if (!x || !y) return x += y;
33    int cur = new_node(), mid = (l+r)>>1;
34    if (l == r) return tr[cur] = tr[x]+tr[y], cur;
35    lc[cur] = merge(l, mid, lc[x], lc[y]);
36    rc[cur] = merge(mid+1, r, rc[x], rc[y]);
37    push_up(cur);
38    return cur;
39  }
40  */
41  void split(int L, int R, int l, int r, int &x, int &y) { //split x
      [L, R] to y
42    if (!x) return;
43    if (L <= l && r <= R) return y = x, x = 0, void();
44    if (!y) y = new_node();
45    int mid = (l+r)>>1;
46    if (L <= mid) split(L, R, l, mid, lc[x], lc[y]);
47    if (R >  mid) split(L, R, mid+1, r, rc[x], rc[y]);
48    push_up(x); push_up(y);
49  }
50  T query(int L, int R, int l, int r, int i) {
51    if (!i) return 0;
52    if (L <= l && r <= R) return tr[i];
53    int mid = (l+r)>>1;
54    if (R <= mid) return query(L, R, l, mid, lc[i]);
55    if (L >  mid) return query(L, R, mid+1, r, rc[i]);
56    return giao(query(L, R, l, mid, lc[i]), query(L, R, mid+1, r, rc[
        i]));
57  }
58  int query_kth(int k, int l, int r, int i) {
59    if (l == r) return l;
60    int mid = (l+r)>>1;
61    if (k <= tr[lc[i]]) return query_kth(k, l, mid, lc[i]);
```

```
    return query_kth(k-tr[lc[i]], mid+1, r, rc[i]);
  }
};
```

21 猫树

所谓”猫树”就是一种不支持修改，仅仅支持快速区间询问的一种静态线段树。

构造一棵这样的静态线段树需要 $O(n\log n)$ 次合并操作，但是此时的查询复杂度被加速至 $O(1)$ 次合并操作。

22 吉老师线段树 | 吉司机线段树

22.1 区间最值操作

1. 区间取 min
2. 询问区间 min
3. 询问区间和

$O(m\log n)$

```
struct Tree {
  struct TreeNode {
    int l, r, mx, se, cnt, tag;
    ll sum;
  } tr[N<<2];
  void push_up(int i) {
    tr[i].sum = tr[i<<1].sum+tr[i<<1|1].sum;
    if (tr[i<<1].mx > tr[i<<1|1].mx) {
      tr[i].mx = tr[i<<1].mx;
      tr[i].cnt = tr[i<<1].cnt;
      tr[i].se = max(tr[i<<1].se, tr[i<<1|1].mx);
    } else if (tr[i<<1].mx < tr[i<<1|1].mx) {
      tr[i].mx = tr[i<<1|1].mx;
      tr[i].cnt = tr[i<<1|1].cnt;
      tr[i].se = max(tr[i<<1|1].se, tr[i<<1].mx);
    } else {
      tr[i].mx = tr[i<<1].mx;
      tr[i].cnt = tr[i<<1].cnt+tr[i<<1|1].cnt;
      tr[i].se = max(tr[i<<1].se, tr[i<<1|1].se);
    }
  }
  void push_tag(int i, int v) {
    if (v >= tr[i].mx) return;
    tr[i].sum -= 1ll*(tr[i].mx-v)*tr[i].cnt;
    tr[i].mx = tr[i].tag = v;
  }
  void push_down(int i) {
    if (tr[i].tag == -1) return;
    push_tag(i<<1, tr[i].tag);
    push_tag(i<<1|1, tr[i].tag);
    tr[i].tag = -1;
  }
  void build(int l, int r, int i = 1) {
    tr[i].l = l; tr[i].r = r; tr[i].tag = -1;
    if (l == r) {
```

```
      tr[i].sum = tr[i].mx = a[l];
      tr[i].cnt = 1;
      tr[i].se = -1;
      return;
    }
    int mid = (l+r)>>1;
    build(l, mid, i<<1);
    build(mid+1, r, i<<1|1);
    push_up(i);
  }
  void update_min(int l, int r, int v, int i = 1) {
    if (v >= tr[i].mx) return;
    if (l <= tr[i].l && r >= tr[i].r && v > tr[i].se) return push_tag
       (i, v);
    push_down(i);
    int mid = (tr[i].l+tr[i].r)>>1;
    if (l <= mid) update_min(l, r, v, i<<1);
    if (r >  mid) update_min(l, r, v, i<<1|1);
    push_up(i);
  }
  int query_max(int l, int r, int i = 1) {
    if (l <= tr[i].l && r >= tr[i].r) return tr[i].mx;
    push_down(i);
    int mid = (tr[i].l+tr[i].r)>>1;
    if (r <= mid) return query_max(l, r, i<<1);
    if (l >  mid) return query_max(l, r, i<<1|1);
    return max(query_max(l, r, i<<1), query_max(l, r, i<<1|1));
  }
  ll query_sum(int l, int r, int i = 1) {
    if (l <= tr[i].l && r >= tr[i].r) return tr[i].sum;
    push_down(i);
    int mid = (tr[i].l+tr[i].r)>>1;
    if (r <= mid) return query_sum(l, r, i<<1);
    if (l >  mid) return query_sum(l, r, i<<1|1);
    return query_sum(l, r, i<<1)+query_sum(l, r, i<<1|1);
  }
};
```

1. 给一个区间 [L,R] 加上一个数 x
2. 把一个区间 [L,R] 里小于 x 的数变成 x
3. 把一个区间 [L,R] 里大于 x 的数变成 x
4. 求区间 [L,R] 的和
5. 求区间 [L,R] 的最大值
6. 求区间 [L,R] 的最小值

$O(m \log^2 n)$

```
struct SegmentTree {
#define rt tr[i]
#define ls tr[i<<1]
#define rs tr[i<<1|1]
  typedef int T;
  struct TreeNode {
    int l, r;
    T mn1, mn2, mx1, mx2, cmn, cmx, tag1, tag2, tag3;
    long long sum;
  };
  vector<T> a;
```

```
12  vector<TreeNode> tr;
13  void push_up(int i) {
14    rt.sum = ls.sum+rs.sum;
15    if (ls.mn1 == rs.mn1) {
16      rt.mn1 = ls.mn1;
17      rt.cmn = ls.cmn+rs.cmn;
18      rt.mn2 = min(ls.mn2, rs.mn2);
19    } else if (ls.mn1 < rs.mn1) {
20      rt.mn1 = ls.mn1;
21      rt.cmn = ls.cmn;
22      rt.mn2 = min(ls.mn2, rs.mn1);
23    } else if (ls.mn1 > rs.mn1) {
24      rt.mn1 = rs.mn1;
25      rt.cmn = rs.cmn;
26      rt.mn2 = min(ls.mn1, rs.mn2);
27    }
28    if (ls.mx1 == rs.mx1) {
29      rt.mx1 = ls.mx1;
30      rt.cmx = ls.cmx+rs.cmx;
31      rt.mx2 = max(ls.mx2, rs.mx2);
32    } else if (ls.mx1 > rs.mx1) {
33      rt.mx1 = ls.mx1;
34      rt.cmx = ls.cmx;
35      rt.mx2 = max(ls.mx2, rs.mx1);
36    } else if (ls.mx1 < rs.mx1) {
37      rt.mx1 = rs.mx1;
38      rt.cmx = rs.cmx;
39      rt.mx2 = max(ls.mx1, rs.mx2);
40    }
41  }
42  // 1 2 3 -> min, max, other
43  void push_tag(int i, T add1, T add2, T add3) {
44    if (rt.mn1 == rt.mx1) {
45      add1 == add3 ? add1 = add2 : add2 = add1; // 不应被其他值的标记
          作用
46      rt.sum += 1ll*rt.cmn*add1;
47    } else {
48      rt.sum += 1ll*rt.cmn*add1+1ll*rt.cmx*add2
49          +(rt.r-rt.l+1ll-rt.cmn-rt.cmx)*add3;
50    }
51    if (rt.mn2 == rt.mx1) rt.mn2 += add2;
52    else if (rt.mn2 != INF) rt.mn2 += add3;
53    if (rt.mx2 == rt.mn1) rt.mx2 += add1;
54    else if (rt.mx2 != -INF) rt.mx2 += add3;
55    rt.mn1 += add1; rt.mx1 += add2;
56    rt.tag1 += add1; rt.tag2 += add2; rt.tag3 += add3;
57  }
58  void push_down(int i) {
59    T mn = min(ls.mn1, rs.mn1);
60    T mx = max(ls.mx1, rs.mx1);
61    push_tag(i<<1  , ls.mn1 == mn ? rt.tag1 : rt.tag3, ls.mx1 == mx ?
        rt.tag2 : rt.tag3, rt.tag3);
62    push_tag(i<<1|1, rs.mn1 == mn ? rt.tag1 : rt.tag3, rs.mx1 == mx ?
        rt.tag2 : rt.tag3, rt.tag3);
63    rt.tag1 = rt.tag2 = rt.tag3 = 0;
64  }
65  template <typename TT> void build(int n, TT arr[]) {
```

```
66      a.resize(1);
67      a.insert(a.end(), arr+1, arr+n+1);
68      tr.resize(n*4+1);
69      build(1, n, 1);
70    }
71    void build(int n, T val = 0) {
72      a.resize(n+1, val);
73      tr.resize(n*4+1);
74      build(1, n, 1);
75    }
76    void build(int l, int r, int i) {
77      rt.l = l; rt.r = r;
78      rt.tag1 = rt.tag2 = rt.tag3 = 0;
79      if (l == r) {
80        rt.sum = rt.mn1 = rt.mx1 = a[l];
81        rt.mn2 = INF; rt.mx2 = -INF;
82        rt.cmn = rt.cmx = 1;
83        return;
84      }
85      int mid = (l+r)>>1;
86      build(l, mid, i<<1); build(mid+1, r, i<<1|1);
87      push_up(i);
88    }
89    void update_add(int l, int r, T v, int i = 1) {
90      if (rt.r < l || rt.l > r) return;
91      if (rt.l >= l && rt.r <= r)
92        return push_tag(i, v, v, v);
93      push_down(i);
94      update_add(l, r, v, i<<1); update_add(l, r, v, i<<1|1);
95      push_up(i);
96    }
97    void update_max(int l, int r, T v, int i = 1) {
98      if (rt.r < l || rt.l > r || rt.mn1 >= v) return;
99      if (rt.l >= l && rt.r <= r && rt.mn2 > v)
100       return push_tag(i, v-rt.mn1, 0, 0);
101     push_down(i);
102     update_max(l, r, v, i<<1); update_max(l, r, v, i<<1|1);
103     push_up(i);
104   }
105   void update_min(int l, int r, T v, int i = 1) {
106     if (rt.r < l || rt.l > r || rt.mx1 <= v) return;
107     if (rt.l >= l && rt.r <= r && rt.mx2 < v)
108       return push_tag(i, 0, v-rt.mx1, 0);
109     push_down(i);
110     update_min(l, r, v, i<<1); update_min(l, r, v, i<<1|1);
111     push_up(i);
112   }
113   long long query_sum(int l, int r, int i = 1) {
114     if (rt.r < l || rt.l > r) return 0;
115     if (rt.l >= l && rt.r <= r) return rt.sum;
116     push_down(i);
117     return query_sum(l, r, i<<1)+query_sum(l, r, i<<1|1);
118   }
119   T query_max(int l, int r, int i = 1) {
120     if (rt.r < l || rt.l > r) return -INF;
121     if (rt.l >= l && rt.r <= r) return rt.mx1;
```

```
    push_down(i);
    return max(query_max(l, r, i<<1), query_max(l, r, i<<1|1));
  }
  T query_min(int l, int r, int i = 1) {
    if (rt.r < l || rt.l > r) return INF;
    if (rt.l >= l && rt.r <= r) return rt.mn1;
    push_down(i);
    return min(query_min(l, r, i<<1), query_min(l, r, i<<1|1));
  }
#undef rt
#undef ls
#undef rs
};
```

22.2 区间历史最值

22.3 区间加区间修改

1. 区间加
2. 区间修改
3. 区间最大值
4. 区间历史最大值

$O(m \log n)$

```
struct SegmentTree {
#define rt tr[i]
#define ls tr[i<<1]
#define rs tr[i<<1|1]
  typedef int T;
  struct TreeNode {
    int l, r;
    bool tag;
    T add, cov, mx, hadd, hcov, hmx;
  } tr[N<<2];
  void push_up(int i) {
    rt.mx = max(ls.mx, rs.mx);
    rt.hmx = max(ls.hmx, rs.hmx);
  }
  void plus(int i, T k, T hk) {
    rt.hmx = max(rt.hmx, rt.mx+hk);
    rt.mx += k;
    rt.tag ? rt.hcov = max(rt.hcov, rt.cov+hk), rt.cov += k
        : rt.hadd = max(rt.hadd, rt.add+hk), rt.add += k;
  }
  void cover(int i, T k, T hk) {
    rt.hmx = max(rt.hmx, hk);
    rt.mx = k;
    rt.hcov = max(rt.hcov, hk);
    rt.cov = k;
    rt.tag = 1;
  }
  void push_down(int i) {
    if (rt.add) {
      plus(i<<1  , rt.add, rt.hadd);
      plus(i<<1|1, rt.add, rt.hadd);
      rt.add = rt.hadd = 0;
```

```
    }
    if (rt.tag) {
      cover(i<<1  , rt.cov, rt.hcov);
      cover(i<<1|1, rt.cov, rt.hcov);
      rt.tag = 0; rt.hcov = -INF;
    }
  }
  void build(int l, int r, int i = 1) {
    rt.l = l; rt.r = r; rt.tag = false;
    rt.add = rt.hadd = 0;
    rt.hcov = -INF;
    if (l == r) return rt.hmx = rt.mx = 0, void();
    int mid = (l+r)>>1;
    build(l, mid, i<<1); build(mid+1, r, i<<1|1);
    push_up(i);
  }
  T query_max(int l, int r, int i = 1) {
    if (rt.r < l || rt.l > r) return -INF;
    if (l <= rt.l && rt.r <= r) return rt.mx;
    push_down(i);
    return max(query_max(l, r, i<<1), query_max(l, r, i<<1|1));
  }
  T query_hmax(int l, int r, int i = 1) {
    if (rt.r < l || rt.l > r) return -INF;
    if (l <= rt.l && rt.r <= r) return rt.hmx;
    push_down(i);
    return max(query_hmax(l, r, i<<1), query_hmax(l, r, i<<1|1));
  }
  void update_add(int l, int r, T v, int i = 1) {
    if (rt.r < l || rt.l > r) return;
    if (l <= rt.l && rt.r <= r) return plus(i, v, v);
    push_down(i);
    update_add(l, r, v, i<<1); update_add(l, r, v, i<<1|1);
    push_up(i);
  }
  void update_cov(int l, int r, T v, int i = 1) {
    if (rt.r < l || rt.l > r) return;
    if (l <= rt.l && rt.r <= r) return cover(i, v, v);
    push_down(i);
    update_cov(l, r, v, i<<1); update_cov(l, r, v, i<<1|1);
    push_up(i);
  }
#undef rt
#undef ls
#undef rs
};
```

22.4 区间最值操作与历史最值询问同向

单点查询

```
struct SegmentTree {
#define rt tr[i]
#define ls tr[i<<1]
#define rs tr[i<<1|1]
  typedef long long T;
```

```
6   struct Tag {
7     T add, mx;
8     Tag(T _add = 0, T _mx = -INF) : add(_add), mx(_mx) {}
9     Tag operator +(const Tag &t) const { // 合并tag
10      return Tag(max(-INF, add+t.add), max(mx+t.add, t.mx));
11    }
12    Tag operator *(const Tag &t) const { // 取max
13      return Tag(max(add, t.add), max(mx, t.mx));
14    }
15    Tag& operator +=(const Tag &t) { return *this = *this+t; }
16    Tag& operator *=(const Tag &t) { return *this = *this*t; }
17  };
18  struct TreeNode {
19    int l, r;
20    Tag his, cur;
21  } tr[N<<2];
22  void push_tag(int i, Tag hk, Tag k) {
23    rt.his *= rt.cur+hk;
24    rt.cur += k;
25  }
26  void push_down(int i) {
27    push_tag(i<<1  , rt.his, rt.cur);
28    push_tag(i<<1|1, rt.his, rt.cur);
29    rt.his = rt.cur = Tag();
30  }
31  void build(int l, int r, int i = 1) {
32    rt.l = l; rt.r = r; rt.his = rt.cur = Tag();
33    if (l == r) return rt.his = rt.cur = Tag(a[l]), void();
34    int mid = (l+r)>>1;
35    build(l, mid, i<<1); build(mid+1, r, i<<1|1);
36  }
37  // add(val, -INF) cov(-INF, val) max(0, val)
38  void update(int l, int r, T a, T x = -INF, int i = 1) {
39    if (rt.r < l || rt.l > r) return;
40    if (l <= rt.l && rt.r <= r) return push_tag(i, Tag(a, x), Tag(a,
        x));
41    push_down(i);
42    update(l, r, a, x, i<<1); update(l, r, a, x, i<<1|1);
43  }
44  T query_max(int x, int i = 1) {
45    if (rt.l == rt.r) return max(rt.cur.add, rt.cur.mx);
46    push_down(i);
47    int mid = (rt.l+rt.r)>>1;
48    if (x <= mid) return query_max(x, i<<1);
49    else return query_max(x, i<<1|1);
50  }
51  T query_hmax(int x, int i = 1) {
52    if (rt.l == rt.r) return max(rt.his.add, rt.his.mx);
53    push_down(i);
54    int mid = (rt.l+rt.r)>>1;
55    if (x <= mid) return query_hmax(x, i<<1);
56    else return query_hmax(x, i<<1|1);
57  }
58  #undef rt
59  #undef ls
60  #undef rs
```

```
};
```

22.5 区间最值操作与历史最值询问反向

1. 区间加
2. 区间 max
3. 询问 min
4. 历史 min

```
struct SegmentTree {
#define rt tr[i]
#define ls tr[i<<1]
#define rs tr[i<<1|1]
  typedef int T;
  struct TreeNode {
    int l, r;
    T mn, hmn, se, tag1, htag1, tag2, htag2;
  } tr[N<<2];
  void push_up(int i) {
    rt.hmn = min(ls.hmn, rs.hmn);
    if (ls.mn == rs.mn) {
      rt.mn = ls.mn;
      rt.se = min(ls.se, rs.se);
    } else if (ls.mn < rs.mn) {
      rt.mn = ls.mn;
      rt.se = min(ls.se, rs.mn);
    } else if (ls.mn > rs.mn) {
      rt.mn = rs.mn;
      rt.se = min(ls.mn, rs.se);
    }
  }
  void push_tag(int i, T add1, T hadd1, T add2, T hadd2) {
    rt.hmn = min(rt.hmn, rt.mn+hadd1);
    rt.htag1 = min(rt.htag1, rt.tag1+hadd1);
    rt.mn += add1; rt.tag1 += add1;
    rt.htag2 = min(rt.htag2, rt.tag2+hadd2);
    if (rt.se != INF) rt.se += add2;
    rt.tag2 += add2;
  }
  void push_down(int i) {
    T mn = min(ls.mn, rs.mn);
    push_tag(i<<1  , ls.mn == mn ? rt.tag1 : rt.tag2,
        ls.mn == mn ? rt.htag1 : rt.htag2, rt.tag2, rt.htag2);
    push_tag(i<<1|1, rs.mn == mn ? rt.tag1 : rt.tag2,
        rs.mn == mn ? rt.htag1 : rt.htag2, rt.tag2, rt.htag2);
    rt.tag1 = rt.htag1 = rt.tag2 = rt.htag2 = 0;
  }
  void build(int l, int r, int i = 1) {
    rt.l = l; rt.r = r;
    rt.tag1 = rt.htag1 = rt.tag2 = rt.htag2 = 0;
    if (l == r) {
      rt.hmn = rt.mn = a[l];
      rt.se = INF;
      return;
    }
    int mid = (l+r)>>1;
```

```
    build(l, mid, i<<1); build(mid+1, r, i<<1|1);
    push_up(i);
  }
  void update_add(int l, int r, T v, int i = 1) {
    if (rt.r < l || rt.l > r) return;
    if (l <= rt.l && rt.r <= r) return push_tag(i, v, v, v, v);
    push_down(i);
    update_add(l, r, v, i<<1); update_add(l, r, v, i<<1|1);
    push_up(i);
  }
  void update_max(int l, int r, T v, int i = 1) {
    if (rt.r < l || rt.l > r || v <= rt.mn) return;
    if (l <= rt.l && rt.r <= r && v < rt.se)
      return push_tag(i, v-rt.mn, v-rt.mn, 0, 0);
    push_down(i);
    update_max(l, r, v, i<<1); update_max(l, r, v, i<<1|1);
    push_up(i);
  }
  T query_min(int l, int r, int i = 1) {
    if (rt.r < l || rt.l > r) return INF;
    if (l <= rt.l && rt.r <= r) return rt.mn;
    push_down(i);
    return min(query_min(l, r, i<<1), query_min(l, r, i<<1|1));
  }
  T query_hmin(int l, int r, int i = 1) {
    if (rt.r < l || rt.l > r) return INF;
    if (l <= rt.l && rt.r <= r) return rt.hmn;
    push_down(i);
    return min(query_hmin(l, r, i<<1), query_hmin(l, r, i<<1|1));
  }
#undef rt
#undef ls
#undef rs
};
```

23 K-D Tree | KDT

1. 将格子 (x,y) 加上 v
2. 求 (xl,yl) 到 (xr,yr) 区间和

```
struct Node {
  int x, y, v;
} s[N];
bool cmpx(int a, int b) { return s[a].x < s[b].x; }
bool cmpy(int a, int b) { return s[a].y < s[b].y; }
struct KDTree {
  double alpha = 0.725;
  int rt, cur, xl, yl, xr, yr; //rt根结点
  int d[N], siz[N], lc[N], rc[N]; //d=1竖着砍, sz子树大小
  int L[N], R[N], D[N], U[N]; //该子树的界线
  int sum[N]; //维护的二维区间信息（二维区间和）
  int g[N], gt;
  void pia(int x) { //将树还原成序列g
    if (!x) return;
```

```
    pia(lc[x]);
    g[++gt] = x;
    pia(rc[x]);
  }
  void push_up(int x) { //更新信息
    siz[x] = siz[lc[x]] + siz[rc[x]] + 1;
    sum[x] = sum[lc[x]] + sum[rc[x]] + s[x].v;
    L[x] = R[x] = s[x].x;
    D[x] = U[x] = s[x].y;
    if (lc[x]) {
      L[x] = min(L[x], L[lc[x]]);
      R[x] = max(R[x], R[lc[x]]);
      D[x] = min(D[x], D[lc[x]]);
      U[x] = max(U[x], U[lc[x]]);
    }
    if (rc[x]) {
      L[x] = min(L[x], L[rc[x]]);
      R[x] = max(R[x], R[rc[x]]);
      D[x] = min(D[x], D[rc[x]]);
      U[x] = max(U[x], U[rc[x]]);
    }
  }
  int build(int l, int r) { //以序列g[l..r]为模板重建树, 返回根结点
    if (l > r) return 0;
    int mid = (l + r) >> 1;
    double ax = 0, ay = 0, sx = 0, sy = 0;
    for (int i = l; i <= r; i++) ax += s[g[i]].x, ay += s[g[i]].y;
    ax /= (r - l + 1);
    ay /= (r - l + 1);
    for (int i = l; i <= r; i++) {
      sx += (ax - s[g[i]].x) * (ax - s[g[i]].x);
      sy += (ay - s[g[i]].y) * (ay - s[g[i]].y);
    }
    if (sx > sy)
      nth_element(g + l, g + mid, g + r + 1, cmpx), d[g[mid]] = 1;
    else
      nth_element(g + l, g + mid, g + r + 1, cmpy), d[g[mid]] = 2;
    lc[g[mid]] = build(l, mid - 1);
    rc[g[mid]] = build(mid + 1, r);
    push_up(g[mid]);
    return g[mid];
  }
  void rebuild(int &x) {
    gt = 0;
    pia(x);
    x = build(1, gt);
  }
  bool bad(int x) {
    return alpha * siz[x] <= max(siz[lc[x]], siz[rc[x]]);
  }
  void insert(int x, int y, int v) { //在(x,y)处插入元素
    s[++cur] = {x, y, v};
    insert(rt, cur);
  }
  void insert(int &x, int v) {
    if (!x) return push_up(x = v);
    if (d[x]) insert(s[v].x <= s[x].x ? lc[x] : rc[x], v);
```

```
    else insert(s[v].y <= s[x].y ? lc[x] : rc[x], v);
    push_up(x);
    if (bad(x)) rebuild(x);
  }
  int query(int xl, int yl, int xr, int yr) { //查询[x1,x2]×[y1,y2]的区间和
    if (xl > xr) swap(xl, xr);
    if (yl > yr) swap(yl, yr);
    this->xl = xl; this->yl = yl; this->xr = xr; this->yr = yr;
    return query(rt);
  }
  int query(int x) {
    if (!x || xr < L[x] || xl > R[x] || yr < D[x] || yl > U[x])
      return 0;
    if (xl <= L[x] && R[x] <= xr && yl <= D[x] && U[x] <= yr) return
      sum[x];
    int res = 0;
    if (xl <= s[x].x && s[x].x <= xr && yl <= s[x].y && s[x].y <= yr)
      res += s[x].v;
    return query(lc[x]) + query(rc[x]) + res;
  }
};
```

24 树状数组

24.1 一维

单点修改区间查询
区间修改单点查询

```
template <typename T>
struct BinaryIndexedTree {
  int n;
  vector<T> tr;
  void init(const int &n) { this->n = n; tr = vector<T>(n+1, 0); }
  void add(const int &x, const T &v) { for (int i = x; i <= n; i += i&-i) tr[i] += v; }
  void add(const int &x, const int &y, const T &v) { add(x, v); add(y+1, -v); }
  T query(const int &x) { T res = 0; for (int i = x ; i; i -= i&-i) res += tr[i]; return res; }
  T query(const int &x, const int &y) { return query(y)-query(x-1); }
};
```

区间修改区间查询

```
struct BIT{
        static const int SIZE=1e6+5;
        ll bit1[SIZE],bit2[SIZE];
        int limit;
        void init(int n=SIZE-1){
                limit=n;
                for(int i=1;i<=n;i++)bit1[i]=bit2[i]=0;
        }
        BIT(){init();}
        void add(ll *bit,int x,ll v){
```

```
                while(x<=limit)bit[x]+=v,x+=x&-x;
        }
        ll query(ll *bit,int x){
                ll res=0;
                while(x)res+=bit[x],x-=x&-x;
                return res;
        }
        void add(int l,int r,ll v){
                add(bit1,l,v);
                add(bit2,l,v*l);
                add(bit1,r+1,-v);
                add(bit2,r+1,-v*(r+1));
        }
        ll query(int l,int r){
                return (r+1)*query(bit1,r)-query(bit2,r)-l*query(bit1
                   ,l-1)+query(bit2,l-1);
        }
}bit;
```

24.2 二维

24.2.1 单点修改区间查询

```
template <typename T>
struct BIT_2D {
  int n, m;
  T a[N][N], tr[N][N];
  BIT_2D() { memset(tr, 0, sizeof tr); }
  void init(const int &_n, const int &_m) {
    n = _n; m = _m;
    memset(a, 0, sizeof a);
    memset(tr, 0, sizeof tr);
  }
  void add(const int &x, const int &y, const T &k) {
    a[x][y] += k;
    for (int i = x; i <= n; i += i&-i)
      for (int j = y; j <= m; j += j&-j)
        tr[i][j] += k;
  }
  T query(const int &x, const int &y) {
    return a[x][y];
    // return query(x, y, x, y);
  }
  T query(int r1, int c1, int r2, int c2) {
    if (r1 > r2) swap(r1, r2);
    if (c1 > c2) swap(c1, c2);
    return _query(r2, c2)-_query(r1-1, c2)-_query(r2, c1-1)+_query(r1
       -1, c1-1);
  }
  T _query(const int &x, const int &y) {
    T res = 0;
    for (int i = x; i; i -= i&-i)
      for (int j = y; j; j -= j&-j)
        res += tr[i][j];
    return res;
  }
```

```
};
```

25 可持久化数组

```
template <typename T>
struct PersistantArray {
  static const int NN = N*(log2(N)+3);
  int rt[N], ls[NN], rs[NN], val[NN], tot, n;
  void build(const int &n) {
    this->n = n;
    tot = 0;
    rt[0] = build(1, n);
  }
  int build(const int &l, const int &r) {
    int cur = ++tot; assert(tot < NN);
    if (l == r) return val[cur] = a[l], cur;
    int mid = (l+r)>>1;
    ls[cur] = build(l, mid);
    rs[cur] = build(mid+1, r);
    return cur;
  }
  void update(const int &cur, const int &pre, const int &x, const T &
     k) {
    rt[cur] = update(rt[pre], x, k, 1, n);
  }
  int update(const int &pre, const int &x, const T &k, const int &l,
     const int &r) {
    int cur = ++tot; assert(tot < NN);
    if (l == x && r == x) return val[cur] = k, cur;
    ls[cur] = ls[pre]; rs[cur] = rs[pre];
    int mid = (l+r)>>1;
    if (x <= mid) ls[cur] = update(ls[pre], x, k, l, mid);
    else rs[cur] = update(rs[pre], x, k, mid+1, r);
    return cur;
  }
  T query(const int &cur, const int &x) {
    return query(rt[cur], x, 1, n);
  }
  T query(const int &cur, const int &x, const int &l, const int &r) {
    if (l == x && r == x) return val[cur];
    int mid = (l+r)>>1;
    if (x <= mid) return query(ls[cur], x, l, mid);
    return query(rs[cur], x, mid+1, r);
  }
};
```

26 可持久化线段树（主席树）

```
template <typename T>
struct PersistantSegmentTree {
  static const int NN = N*(log2(N)+5);
  int rt[N], sum[NN], ls[NN], rs[NN], tot, n;
```

```
  void build(const int &n) {
    this->n = n;
    tot = 0;
    rt[0] = _build(1, n);
  }
  void update(const int &cur, const int &pre, const T &k) {
    rt[cur] = _update(rt[pre], 1, n, k);
  }
  T query(const int &l, const int &r, const int &k) {
    return _query(rt[l-1], rt[r], 1, n, k);
  }
private:
  int _build(const int &l, const int &r) {
    int cur = ++tot;
    sum[cur] = 0;
    if (l >= r) return cur;
    int mid = (l+r)>>1;
    ls[cur] = _build(l, mid);
    rs[cur] = _build(mid+1, r);
    return cur;
  }
  int _update(const int &pre, const int &l, const int &r, const int &
     k) {
    int cur = ++tot;
    ls[cur] = ls[pre]; rs[cur] = rs[pre]; sum[cur] = sum[pre]+1;
    if (l >= r) return cur;
    int mid = (l+r)>>1;
    if (k <= mid) ls[cur] = _update(ls[pre], l, mid, k);
    else rs[cur] = _update(rs[pre], mid+1, r, k);
    return cur;
  }
  int _query(const int &u, const int &v, const int &l, const int &r,
     const int &k) {
    if (l >= r) return l;
    int num = sum[ls[v]]-sum[ls[u]], mid = (l+r)>>1;
    if (num >= k) return _query(ls[u], ls[v], l, mid, k);
    else return _query(rs[u], rs[v], mid+1, r, k-num);
  }
};
```

动态开点?oj 可以跑本地莫名 re

```
template <class T>
class PersistantSegmentTree {
 public:
  void init(int n) {
    tot = 0;
    this->n = n;
    rt = vector<int>(1, 0);
    tr = vector<TreeNode>(1, TreeNode{0, 0, 0});
  }
  void update(int cur, int pre, int k, T v = 1) {
    rt.resize(cur+1); // assert rt.size()+1 <= cur
    rt[cur] = update(rt[pre], 1, n, k, v);
  }
  int update(int pre, int l, int r, int k, T v) {
    int cur = ++tot; // assert tot == tr.size()
    tr.emplace_back(tr[pre]);
```

```
    tr[cur].sum += v;
    if (l == r) return cur;
    int mid = (l+r)>>1;
    if (k <= mid) tr[cur].lc = update(tr[pre].lc, l, mid, k, v);
    else tr[cur].rc = update(tr[pre].rc, mid+1, r, k, v);
    return cur;
  }
  T query(int l, int r, int ql, int qr) {
    return query(rt[l-1], rt[r], 1, n, ql, qr);
  }
  // [u, v] 段 [ql, qr] 区间和
  T query(int u, int v, int l, int r, int ql, int qr) {
    if (!u && !v) return 0;
    if (l >= ql && r <= qr) return tr[v].sum-tr[u].sum;
    int mid = (l+r)>>1;
    if (qr <= mid) return query(tr[u].lc, tr[v].lc, l, mid, ql, qr);
    if (ql >  mid) return query(tr[u].rc, tr[v].rc, mid+1, r, ql, qr)
       ;
    return query(tr[u].lc, tr[v].lc, l, mid, ql, qr)+query(tr[u].rc,
       tr[v].rc, mid+1, r, ql, qr);
  }
 private:
  struct TreeNode {
    int lc, rc;
    T sum;
  };
  int tot, n;
  vector<int> rt;
  vector<TreeNode> tr;
};
```

dingbacode 高地

```
template <typename T>
struct PersistantSegmentTree {
  static const int NN = N*(log2(N)+5);
  int rt[N], ls[NN], rs[NN], tot, n;
  T sum[NN];
  void build(const int &n) {
    this->n = n;
    tot = 0;
    rt[0] = _build(1, n);
  }
  void update(const int &cur, const int &pre, const int &k, const T &
     v) {
    rt[cur] = _update(rt[pre], 1, n, k, v);
  }
  T query(const int &l, const int &r, const int &ql, const int &qr) {
    return _query(rt[l-1], rt[r], 1, n, ql, qr);
  }
private:
  int _build(const int &l, const int &r) {
    int cur = ++tot;
    sum[cur] = 0;
    if (l >= r) return cur;
    int mid = (l+r)>>1;
    ls[cur] = _build(l, mid);
    rs[cur] = _build(mid+1, r);
```

```
    return cur;
  }
  int _update(const int &pre, const int &l, const int &r, const int &k, const T &v) {
    int cur = ++tot;
    ls[cur] = ls[pre]; rs[cur] = rs[pre]; sum[cur] = sum[pre]+v;
    if (l >= r) return cur;
    int mid = (l+r)>>1;
    if (k <= mid) ls[cur] = _update(ls[pre], l, mid, k, v);
    else rs[cur] = _update(rs[pre], mid+1, r, k, v);
    return cur;
  }
  // [u, v] 段 [ql, qr] 区间和
  T _query(const int &u, const int &v, const int &l, const int &r, const int &ql, const int &qr) {
    if (l >= ql && r <= qr) return sum[v]-sum[u];
    int mid = (l+r)>>1;
    if (qr <= mid) return _query(ls[u], ls[v], l, mid, ql, qr);
    if (ql >  mid) return _query(rs[u], rs[v], mid+1, r, ql, qr);
    return _query(ls[u], ls[v], l, mid, ql, qr)+_query(rs[u], rs[v], mid+1, r, ql, qr);
  }
};
```

27 分块

例题

```
struct FenKuai {
  typedef long long T;
  int t; // 每组大小
  static const int NN = static_cast<int>(sqrt(N))+7;
  T a[N], sum[NN], add[NN];
  FenKuai() {
    memset(a, 0, sizeof a);
    memset(sum, 0, sizeof sum);
    memset(add, 0, sizeof add);
  }
  void init() {
    t = static_cast<int>(sqrt(n)+0.5);
    for (int i = 0; i < n; ++i) sum[i/t] += a[i];
  }
  void update(int x, T k) { a[x] += k; sum[x/t] += k; }
  void update(int x, int y, T k) {
    for ( ; x <= y && x%t; ++x) a[x] += k, sum[x/t] += k;
    for ( ; x+t-1 <= y; x += t) sum[x/t] += k*t, add[x/t] += k;
    for ( ; x <= y; ++x) a[x] += k, sum[x/t] += k;
  }
  T query(int x) { return a[x]+add[x/t]; }
  T query(int x, int y) {
    T res = 0;
    for ( ; x <= y && x%t; ++x) res += a[x]+add[x/t];
    for ( ; x+t-1 <= y; x += t) res += sum[x/t];
    for ( ; x <= y; ++x) res += a[x]+add[x/t];
    return res;
  }
```

```
} B;
```

```
struct FenKuai {
  typedef int T;
  int t; // 每组大小
  T a[N], b[N], add[N];
  FenKuai() {
    memset(a, 0, sizeof a);
    memset(b, 0, sizeof b);
    memset(add, 0, sizeof add);
  }
  void build(int x) {
    for (int i = x*t; i < min(x*t+t, n); ++i) b[i] = a[i];
    sort(b+x*t, b+min(x*t+t, n));
  }
  void init() {
    t = static_cast<int>(sqrt(n)+0.5);
    for (int i = 0; i*t < n; ++i) build(i);
  }
  void update(int x, int y, T c) {
    int i = x;
    for ( ; i <= y && i%t; ++i) a[i] += c;
    build(x/t);
    for ( ; i+t-1 <= y; i += t) add[i/t] += c;
    for ( ; i <= y; ++i) a[i] += c;
    build(y/t);
  }
  T query(int x, int y, long long c) {
    T res = 0; int i = x;
    for ( ; i <= y && i%t; ++i) res += (a[i]+add[i/t] < c*c);
    for ( ; i+t-1 <= y; i += t) res += lower_bound(b+i, b+i+t, c*c-
       add[i/t])-(b+i);
    for ( ; i <= y; ++i) res += (a[i]+add[i/t] < c*c);
    return res;
  }
} B;
```

```
struct FenKuai {
  typedef int T;
  int t, sz;
  static const int NN = static_cast<int>(sqrt(N))+7;
  T a[N];
  deque<int> q[NN];
  void init(int _n) {
    sz = _n;
    t = static_cast<int>(sqrt(sz*1.5)+0.5);
    for (int i = 0; i < sz; ++i) q[i/t].push_back(a[i]);
  }
  void update(int x, int k) {
    stack<int> tmp;
    for (int i = 0; i != x%t; ++i) {
      tmp.push(q[x/t].front());
      q[x/t].pop_front();
    }
    q[x/t].push_front(k);
    while (tmp.size()) {
      q[x/t].push_front(tmp.top());
```

```
      tmp.pop();
    }
    ++sz;
    if (sz/t == x/t) return;
    for (int i = x/t, val; i < sz/t; ++i) {
      val = q[i].back();
      q[i].pop_back();
      q[i+1].push_front(val);
    }
  }
  T query(int x) {
    stack<int> tmp;
    for (int i = 0; i != x%t; ++i) {
      tmp.push(q[x/t].front());
      q[x/t].pop_front();
    }
    int res = q[x/t].front();
    while (tmp.size()) {
      q[x/t].push_front(tmp.top());
      tmp.pop();
    }
    return res;
  }
} B;
```

28 莫队

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

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

28.1 奇偶性排序

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

28.2 带修改莫队

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

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

28.3 值域分块

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

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

template <typename T>
struct SufSum {
  int n, block;
  T s[N], t[(int)sqrt(N)+3];
  void init(int n) {
    this->n = n;
    block = sqrt(n);
  }
  void add(int x, T k) {
    for (int i = x; i/block == x/block && i >= 1; --i) s[i] += k;
    for (int i = x/block-1; i >= 0; --i) t[i] += k;
  }
  T query(int x) {
    return t[x/block]+s[x];
  }
};
```

28.4 二次离线莫队

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

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

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

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

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

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

对于 $(1,i-1)(i,i)$ 没什么好说，暴力处理前缀和

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

求区间逆序对

```
struct Query {
  int id, l, r;
  Query() {}
  Query(int i, int _l, int _r) : id(i), l(_l), r(_r) {}
};

int n, m, block;
int a[N];
long long res[N], sumil[N], sumir[N], ans[N];
Query q[N];
SufSum<int> suml;
PreSum<int> sumr;
vector<Query> ql[N], qr[N];

inline void calc_sumi() {
  static BinaryIndexedTree<int> tree;
  tree.init(n);
  for (int i = 1; i <= n; ++i) {
    sumil[i] = sumil[i-1]+i-1-tree.query(a[i]);
    tree.add(a[i], 1);
  }
  tree.clear();
  for (int i = n; i; --i) {
    sumir[i] = sumir[i+1]+tree.query(a[i]-1);
    tree.add(a[i], 1);
  }
}

signed main() {
  read(n); read(m);
  for (int i = 1; i <= n; ++i) read(a[i]);
  discrete();
  block = n/sqrt(m);
  for (int i = 1; i <= m; ++i) {
    q[i].id = i;
    read(q[i].l);
    read(q[i].r);
  }
  sort(q+1, q+m+1, cmp);
  calc_sumi();
  q[0] = Query(0, 1, 0);
  for (int i = 1, ul, vl, ur, vr; i <= m; ++i) {
    ul = q[i-1].l; ur = q[i-1].r;
    vl = q[i].l; vr = q[i].r;
    res[i] = sumil[vr]-sumil[ur]+sumir[vl]-sumir[ul];
    if (vl < ul) qr[vr+1].emplace_back(-i, vl, ul-1);
    if (vl > ul) qr[vr+1].emplace_back(+i, ul, vl-1);
    if (vr < ur) ql[ul-1].emplace_back(+i, vr+1, ur);
    if (vr > ur) ql[ul-1].emplace_back(-i, ur+1, vr);
  }
  suml.init(n+1);
  for (int i = 1; i <= n; ++i) {
    suml.add(a[i], 1);
    for (auto &qq : ql[i]) {
      for (int j = qq.l; j <= qq.r; ++j) {
        if (qq.id > 0) res[qq.id] += suml.query(a[j]+1);
```

```
57          else res[-qq.id] -= suml.query(a[j]+1);
58        }
59      }
60    }
61    sumr.init(n);
62    for (int i = n; i; --i) {
63      sumr.add(a[i], 1);
64      for (auto &qq : qr[i]) {
65        for (int j = qq.l; j <= qq.r; ++j) {
66          if (qq.id > 0) res[qq.id] += sumr.query(a[j]-1);
67          else res[-qq.id] -= sumr.query(a[j]-1);
68        }
69      }
70    }
71    for (int i = 1; i <= m; ++i) {
72      res[i] += res[i-1];
73      ans[q[i].id] = res[i];
74    }
75    for (int i = 1; i <= m; ++i) write(ans[i]), putchar('\n');
76    return 0;
77  }
```

29 ST 表

29.1 一维

```
1   template <typename T, typename U = std::greater<T>>
2   struct ST {
3     static const int NN = 31-__builtin_clz(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  };
```

29.2 二维

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

```
template <typename T, typename U = std::greater<T>>
struct ST {
  static const int NN = (int)log2(N)+3;
  static const T INF = 1e9;
  U cmp = U();
  T rmq[N][N][NN][NN]; // rmq[i][j][k][l] [i, j] [i+2^k-1, j+2^l-1]
  ST() { init(); }
  ST(const T &val) { init(val); }
  T& operator [] (const int &i) { return rmq[i][0]; }
  void init(){ fill(rmq[0][0][0], rmq[0][0][0]+N*N*NN*NN, cmp(-INF, +INF) ? INF : -INF); }
  void init(const T &val) { fill(rmq[0][0][0], rmq[0][0][0]+N*N*NN*NN, val); }
  T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
  void build(T a[N][N], const int &n, const int &m) {
    for (int k = 0; k <= log_2[n]; ++k)
    for (int l = 0; l <= log_2[m]; ++l)
    for (int i = 1; i+(1<<k)-1 <= n; ++i)
    for (int j = 1; j+(1<<l)-1 <= m; ++j) {
      T &cur = rmq[i][j][k][l];
      if (!k && !l) cur = a[i][j];
      else if (!l) cur = mv(rmq[i][j][k-1][l], rmq[i+(1<<(k-1))][j][k-1][l]);
      else cur = mv(rmq[i][j][k][l-1], rmq[i][j+(1<<(l-1))][k][l-1]);
    }
  }
  T query(const int &r1, const int &c1, const int &r2, const int &c2) {
    int k = log_2[r2-r1+1], l = log_2[c2-c1+1];
    return mv(mv(rmq[r1][c1][k][l], rmq[r2-(1<<k)+1][c2-(1<<l)+1][k][l]),
        mv(rmq[r2-(1<<k)+1][c1][k][l], rmq[r1][c2-(1<<l)+1][k][l]));
  }
};
```

29.3 反向 ST

```
template <typename T, typename U = std::greater<T>>
struct rST {
  static const int NN = (int)log2(N)+3;
  static const T INF = 1e9;
  int n;
  int lg2[N];
  U cmp = U();
  T rmq[N][NN]; // rmq[i][j] ==> [i, i+2^j-1]
  rST() { for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1; }
  T& operator [] (const int &i) { return rmq[i][0]; }
  T mv(const T &x, const T &y) { return cmp(x, y) ? x : y; }
  void init(const int &_n, const T &val = 0) {
    n = _n;
    for (int i = 1; i <= n; ++i) fill(rmq[i], rmq[i]+NN, val);
```

```
  }
  void update(const int &l, const int &r, const T &k) {
    if (l > r) return void(update(r, l, k));
    int b = lg2[r-l+1];
    rmq[l][b] = mv(rmq[l][b], k);
    rmq[r-(1<<b)+1][b] = mv(rmq[r-(1<<b)+1][b], k);
  }
  void build() {
    for (int i = lg2[n]; i >= 0; --i) {
      for (int l = 1, r; l <= n; ++l) {
        r = l+(1<<i);
        if (r <= n) rmq[r][i] = mv(rmq[r][i], rmq[l][i+1]);
        rmq[l][i] = mv(rmq[l][i], rmq[l][i+1]);
      }
    }
  }
  T query(const int &l, const int &r) {
    if (l > r) return query(r, l);
    int b = lg2[r-l+1];
    return mv(rmq[l][b], rmq[r-(1<<b)+1][b]);
  }
};
```

30 并查集

30.1 路径压缩

```
struct DSU {
  vector<int> fa;
  void init(int n) { fa = vector<int>(n+1); iota(fa.begin(), fa.end()
     , 0); }
  int get(int s) { return s == fa[s] ? s : fa[s] = get(fa[s]); }
  int& operator [] (int i) { return fa[get(i)]; }
  bool merge(int x, int y) { // merge x to y
    x = get(x); y = get(y);
    return x == y ? false : (fa[x] = y, true);
  }
};
```

30.2 按秩合并

秩的意思就是树的高度，按秩合并过后并查集的结构为树形结构，最坏情况为 $O(m \log n)$

```
struct DSU {
  vector<int> fa, rk;
  void init(int n) { fa = rk = vector<int>(n+1, 0); iota(fa.begin(),
     fa.end(), 0); }
  int get(int s) { return s == fa[s] ? s : get(fa[s]); }
  int operator ; }
  bool merge(int x, int y) {
    x = get(x); y = get(y);
    if (x == y) return false;
    if (rk[x] < rk[y]) fa[x] = y;
    else fa[y] = x, rk[x] += rk[x] == rk[y];
```

```
    return true;
  }
};
```

30.3 带权并查集

```
template <typename T = int> struct DSU {
  vector<int> fa;
  vector<T> w;
  void init(int n, T v = 1) {
    fa = vector<int>(n+1);
    iota(fa.begin(), fa.end(), 0);
    w = vector<T>(n+1, v);
  }
  void init(int n, T a[]) {
    fa = vector<int>(n+1);
    iota(fa.begin(), fa.end(), 0);
    w = vector<T>(a, a+n+1);
  }
  int get(int s) { return s == fa[s] ? s : fa[s] = get(fa[s]); }
  int& operator [] (int i) { return fa[get(i)]; }
  bool merge(int x, int y) { // merge x to y
    x = get(x); y = get(y);
    return x == y ? false : (w[y] += w[x], fa[x] = y, true);
  }
};
```

30.4 扩展域并查集

例题：关押罪犯，食物链

```
struct DSU {
  int n;
  vector<int> fa; // [1, n] partner, [n+1, 2n] enemy
  void init(int n) {
    this->n = n;
    fa = vector<int>(2*n+1, 0);
    iota(fa.begin(), fa.end(), 0);
  }
  int get(int s) { return s == fa[s] ? s : fa[s] = get(fa[s]); }
  int& operator [] (int i) { return fa[get(i)]; }
  void merge(int x, int y) { fa[get(x)] = get(y); }
  bool update(int x, int y) {
    if (get(x) == get(y)) return false;
    merge(x+n, y);
    merge(x, y+n);
    return true;
  }
};
```

30.5 可撤销并查集

用一个栈维护每次操作

30.5.1 按秩合并

```
struct DSU {
  vector<int> fa, rk;
  stack<pair<int&, int>> stk;
  void init(int n) {
    stk = stack<pair<int&, int>>();
    fa = rk = vector<int>(n+1, 0);
    iota(fa.begin(), fa.end(), 0);
  }
  int get(int s) { while (s != fa[s]) s = fa[s]; return s; }
  int& operator ]; }
  int merge(int x, int y) { // return the number push in stack
    x = get(x); y = get(y);
    if (x == y) return 0;
    if (rk[x] > rk[y]) swap(x, y);
    stk.push({fa[x], fa[x]});
    fa[x] = y;
    return rk[x] == rk[y] ? stk.push({rk[y], rk[y]++}), 2 : 1;
  }
  bool undo() {
    if (stk.empty()) return false;
    stk.top().first = stk.top().second;
    stk.pop();
    return true;
  }
};
```

30.5.2 启发式合并

```
struct DSU {
  vector<int> fa, sz;
  stack<pair<int, int>> stk;
  void init(int n) {
    stk = stack<pair<int, int>>();
    fa = sz = vector<int>(n+1, 1);
    iota(fa.begin(), fa.end(), 0);
  }
  int get(int s) { while (s != fa[s]) s = fa[s]; return s; }
  int& operator ]; }
  int merge(int x, int y) {
    x = get(x); y = get(y);
    if (x == y) return 0;
    if (sz[x] > sz[y]) swap(x, y);
    stk.push({x, y});
    fa[x] = y;
    sz[y] += sz[x];
    return 1;
  }
  bool undo() {
    if (stk.empty()) return false;
    int x = stk.top().first, y = stk.top().y;
    stk.pop();
    fa[x] = x;
    sz[y] -= sz[x];
```

```
    return true;
  }
}
```

30.5.3 可撤销扩展域并查集

```
struct DSU {
  int n;
  vector<int> fa, rk; // [1, n] partner, [n+1, 2n] enemy
  stack<pair<int&, int>> stk;
  void init(int n) {
    this->n = n;
    stk = stack<pair<int&, int>>();
    fa = rk = vector<int>(2*n+1, 0);
    iota(fa.begin(), fa.end(), 0);
  }
  int& operator [] (int i) { return fa[get(i)]; }
  int get(int s) { while (s != fa[s]) s = fa[s]; return s; }
  void undo() { stk.top().first = stk.top().second; stk.pop(); }
  void merge(int x, int y) {
    x = get(x); y = get(y);
    if (x == y) return;
    if (rk[x] > rk[y]) swap(x, y);
    stk.push({fa[x], fa[x]});
    stk.push({rk[y], rk[y]});
    fa[x] = y;
    rk[y] += rk[x] == rk[y];
  }
  bool update(int x, int y) {
    if (get(x) == get(y)) return false;
    merge(x+n, y);
    merge(x, y+n);
    return true;
  }
};
```

30.6 可持久化并查集

```
struct PersistantUnionSet {
  static const int NN = N*(log2(N)+3);
  int rt[N], ls[NN], rs[NN], fa[NN], dep[NN], n, tot;
  void build(const int &n) {
    this->n = n;
    tot = 0;
    rt[0] = build(1, n);
  }
  int build(const int &l, const int &r) {
    int cur = ++tot; assert(tot < NN);
    if (l == r) return fa[cur] = l, dep[cur] = 0, cur;
    int mid = (l+r)>>1;
    ls[cur] = build(l, mid);
    rs[cur] = build(mid+1, r);
    return cur;
  }
```

```
17    bool query(const int &cur, const int &x, const int &y) {
18      return fa[getf(rt[cur], x)] == fa[getf(rt[cur], y)];
19    }
20    // return the id of fa[], dep[]
21    int query(const int &cur, const int &x, const int &l, const int &r)
          {
22      if (l == r) return cur;
23      int mid = (l+r)>>1;
24      if (x <= mid) return query(ls[cur], x, l, mid);
25      else return query(rs[cur], x, mid+1, r);
26    }
27    // return the id of fa[], dep[]
28    int getf(const int &cur, int x) {
29      int fi;
30      while (fa[(fi = query(cur, x, 1, n))] != x) x = fa[fi];
31      return fi;
32    }
33    void merge(const int &cur, const int &pre, const int &x, const int
          &y) {
34      rt[cur] = rt[pre];
35      int fx = getf(rt[cur], x), fy = getf(rt[cur], y);
36      if (fa[fx] == fa[fy]) return;
37      if (dep[fx] > dep[fy]) swap(fx, fy);
38      rt[cur] = update(rt[pre], fa[fx], fa[fy], 1, n);
39      if (dep[fx] == dep[fy]) add(rt[cur], fa[fy], 1, n);
40    }
41    // update fa, merge x to y
42    int update(const int &pre, const int &x, const int &y, const int &l
          , const int &r) {
43      int cur = ++tot; assert(tot < NN);
44      if (l == r) return fa[cur] = y, dep[cur] = dep[pre], cur;
45      ls[cur] = ls[pre]; rs[cur] = rs[pre];
46      int mid = (l+r)>>1;
47      if (x <= mid) ls[cur] = update(ls[pre], x, y, l, mid);
48      else rs[cur] = update(rs[pre], x, y, mid+1, r);
49      return cur;
50    }
51    // add dep
52    void add(const int &cur, const int &x, const int &l, const int &r)
          {
53      if (l == r) return ++dep[cur], void();
54      int mid = (l+r)>>1;
55      if (x <= mid) add(ls[cur], x, l, mid);
56      else add(rs[cur], x, mid+1, r);
57    }
58  };
```

31 左偏树 | 可并堆

1 x y：将第 x 个数和第 y 个数所在的小根堆合并（若第 x 或第 y 个数已经被删除或第 x 和第 y 个数在用一个堆内，则无视此操作）。

2 x：输出第 x 个数所在的堆最小数，并将这个最小数删除（若有多个最小数，优先删除先输入的；若第 x 个数已经被删除，则输出 -1 并无视删除操作）。

```
1   template <typename T> struct Tree { // 左偏树|可并堆
```

```
#define ls tr[x].son[0]
#define rs tr[x].son[1]
  struct TreeNode {
    T val;
    int dis, rt, son[2];
  };
  vector<TreeNode> tr;
  template <typename TT> void build(TT a[], int n) {
    tr.resize(n+1);
    tr[0].dis = -1;
    for (int i = 1; i <= n; ++i) {
      tr[i].val = a[i];
      tr[i].rt = i;
    }
  }
  int get(int x) {
    return tr[x].rt == x ? x : tr[x].rt = get(tr[x].rt);
  }
  void merge(int x, int y) {
    if (tr[x].val == -1 || tr[y].val == -1) return;
    x = get(x); y = get(y);
    if (x != y) tr[x].rt = tr[y].rt = _merge(x, y);
  }
  int _merge(int x, int y) {
    if (!x || !y) return x+y;
    if (tr[x].val > tr[y].val || (tr[x].val == tr[y].val && x > y))
       swap(x, y);
    rs = _merge(rs, y);
    if(tr[ls].dis < tr[rs].dis) swap(ls, rs);
    tr[ls].rt = tr[rs].rt = tr[x].rt = x;
    tr[x].dis = tr[rs].dis+1;
    return x;
  }
  T pop(int x) {
    if (tr[x].val == -1) return -1;
    x = get(x);
    T res = tr[x].val;
    tr[x].val = -1;
    tr[ls].rt = ls;
    tr[rs].rt = rs;
    tr[x].rt = _merge(ls, rs);
    return res;
  }
#undef ls
#undef rs
};
```

Part IV
字符串

32 回文字符串 |manacher 算法

从 0 开始，第 i 位对应 p[i*2+2]

```
inline int manacher(const char *str, char *buf, int *p) {
  int str_len = strlen(str), buf_len = 2;
  buf[0] = buf[1] = '#';
  for(int i = 0; i < str_len; ++i)
    buf[buf_len++] = str[i], buf[buf_len++] = '#';

  int mx = 0, id, ans = 0;
  for(int i = 1; i < buf_len; ++i) {
    if(i <= mx) p[i] = min(p[id*2-i], mx-i);
    else p[i] = 1;
    while(buf[i-p[i]] == buf[i+p[i]]) p[i]++;
    if(i+p[i] > mx) mx = i+p[i], id = i;
    ans = max(ans, p[i]-1);
  }
  return ans;
}
```

32.1 判断 s[l, r] 是否为回文

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

33 KMP

```
inline void get_next(const char *s, int nex[]) {
  nex[0] = nex[1] = 0;
  for (int i = 1, j = 0, l = strlen(s); i < l; ++i) {
    while (j && s[i] != s[j]) j = nex[j];
    nex[i+1] = s[i] == s[j] ? ++j : 0;
  }
}

inline void kmp(const char *s1, const char *s2, int nex[]) {
  for (int i = 0, j = 0, l1 = strlen(s1), l2 = strlen(s2); i < l1; ++
     i){
    while (j && s1[i] != s2[j]) j = nex[j];
    if (s1[i] == s2[j]) ++j;
    if (j == l2) {
      cout << i-l2+2 << endl;
      j = nex[j];
    }
  }
}
```

nex 数组往上跳为公差为 i-nex[i] 的等差数列直到 i/2

34 扩展 KMP|Z 函数

```
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
```

```
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

35 字典树

```
struct TireTree {
  static const int NN = 5e5+7;
  static const int SZ = 26;
  char beg;
  int nex[NN][SZ], num[NN], cnt;
  bool exist[NN];
  TireTree(char _beg = 'a') : beg(_beg) { clear(); }
  void clear() {
    memset(nex, 0, sizeof(nex[0])*(cnt+1));
    memset(num, 0, sizeof(int)*(cnt+1));
    memset(exist, 0, sizeof(bool)*(cnt+1));
    cnt = 0;
  }
  void insert(const char *s) {
    int len = strlen(s), p = 0;
    for (int i = 0, c; i < len; ++i) {
      c = s[i]-beg;
      if (!nex[p][c]) nex[p][c] = ++cnt;
      p = nex[p][c];
      ++num[p];
    }
    exist[p] = true;
  }
  bool find(const char *s) {
    int len = strlen(s), p = 0;
    for (int i = 0, c; i < len; ++i) {
      c = s[i]-beg;
      if (!nex[p][c]) return false;
      p = nex[p][c];
    }
    return exist[p];
  }
  int count(const char *s) {
    int len = strlen(s), p = 0;
    for (int i = 0, c; i < len; ++i) {
      c = s[i]-beg;
      if (!nex[p][c]) return 0;
      p = nex[p][c];
    }
    return num[p];
  }
```

```
  void insert(const string &s) { insert(s.c_str()); }
  bool find(const string &s) { return find(s.c_str()); }
  int count(const string &s) { return count(s.c_str()); }
};
```

35.1 求异或

最大异或

```
struct TireTree {
  static const int SZ = 2;
  static const int B = 30;
  static const int NN = N*B;
  int nex[NN][SZ], cnt;
  void init() { clear(); }
  void clear() {
    memset(nex, 0, sizeof(int)*(cnt+1)*SZ);
    cnt = 0;
  }
  void insert(int x) {
    for (int i = B, c, p = 0; i >= 0; --i) {
      c = (x>>i)&1;
      if (!nex[p][c]) nex[p][c] = ++cnt;
      p = nex[p][c];
    }
  }
  int max_xor(int x) const {
    int ans = 0;
    for (int i = B, c, p = 0; i >= 0; --i) {
      c = (x>>i)&1;
      if (!nex[p][c^1]) p = nex[p][c];
      else p = nex[p][c^1], ans |= 1<<i;
    }
    return ans;
  }
};
```

动态开点 + 支持合并

```
struct TireTree {
  static const int SZ = 2;
  static const int B = 30;
  typedef array<int, SZ> T;
  vector<T> nex;
  TireTree() { init(); }
  void init() { nex.assign(1, T()); /* nex.reserve(N); */ }
  void clear() { nex = vector<T>(); }
  size_t size() const { return nex.size(); }
  void extend(int &x) {
    if (x != 0) return;
    x = nex.size();
    nex.emplace_back(T());
  }
  void insert(int x) {
    for (int i = B, c, p = 0; i >= 0; --i) {
      c = (x>>i)&1;
      extend(nex[p][c]);
```

```
      p = nex[p][c];
    }
  }
  int max_xor(int x) const {
    int ans = 0;
    for (int i = B, c, p = 0; i >= 0; --i) {
      c = (x>>i)&1;
      if (!nex[p][c^1]) p = nex[p][c];
      else p = nex[p][c^1], ans |= 1<<i;
    }
    return ans;
  }
  void dfs(const TireTree &t, int p = 0, int pt = 0) {
    for (int c = 0; c < SZ; ++c) {
      if (t.nex[pt][c] == 0) continue;
      extend(nex[p][c]);
      dfs(t, nex[p][c], t.nex[pt][c]);
    }
  }
  void join(TireTree &t) {
    if (t.size() < size()) swap(*this, t);
    dfs(t);
    t.clear();
  }
};
```

36 AC 自动机

AC 自动机是以 Trie 的结构为基础，结合 KMP 的思想建立的。
将所有模式串构成一棵 Trie，再对所有结点构造失配指针
如需构造可重建 AC 自动机，每次构造建一个 nex 数组的拷贝

```
struct Aho_Corasick_Automaton {
  static const int NN = 5e6+7;
  static const int SZ = 26;
  char beg;
  int nex[NN][SZ], num[NN], fail[NN], cnt;
  Aho_Corasick_Automaton(const char &_beg = 'a') : beg(_beg) {}
  void clear() {
    memset(nex, 0, sizeof(nex[0])*(cnt+1));
    memset(num, 0, sizeof(int)*(cnt+1));
    memset(fail, 0, sizeof(int)*(cnt+1));
    cnt = 0;
  }
  void insert(const char *s) {
    int len = strlen(s), p = 0;
    for (int i = 0, c; i < len; ++i) {
      c = s[i]-beg;
      if (!nex[p][c]) nex[p][c] = ++cnt;
      p = nex[p][c];
    }
    ++num[p];
  }
  void build() {
    static queue<int> q;
    for (int i = 0; i < SZ; ++i) if (nex[0][i]) q.push(nex[0][i]);
```

```
    while (q.size()) {
      int u = q.front();
      q.pop();
      for (int i = 0; i < SZ; ++i) {
        if (nex[u][i]) {
          fail[nex[u][i]] = nex[fail[u]][i];
          q.push(nex[u][i]);
        } else {
          nex[u][i] = nex[fail[u]][i];
        }
      }
    }
  }
  int query(const char *s) {
    int len = strlen(s), p = 0, res = 0;
    for (int i = 0; i < len; ++i) {
      p = nex[p][s[i]-beg];
      for (int t = p; t && ~num[t]; t = fail[t]) {
        res += num[t];
        num[t] = -1;
      }
    }
    return res;
  }
};
```

```
struct Aho_Corasick_Automaton {
  static const int NN = 2e5+7;
  static const int SZ = 26;
  char beg;
  int cnt;
  int nex[NN][SZ], fail[NN], vis[NN];
  Aho_Corasick_Automaton(const char &_beg = 'a') : beg(_beg) {}
  void clear() {
    memset(nex, 0, sizeof(nex[0])*(cnt+1));
    memset(fail, 0, sizeof(int)*(cnt+1));
    memset(vis, 0, sizeof(int)*(cnt+1));
    cnt = 0;
  }
  int insert(const char *s) {
    int len = strlen(s), p = 0;
    for (int i = 0, c; i < len; ++i) {
      c = s[i]-beg;
      if (!nex[p][c]) nex[p][c] = ++cnt;
      p = nex[p][c];
    }
    return p;
  }
  void build() {
    static queue<int> q;
    for (int i = 0; i < SZ; ++i) if (nex[0][i]) q.push(nex[0][i]);
    while (q.size()) {
      int u = q.front();
      q.pop();
      for (int i = 0; i < SZ; ++i) {
        if (nex[u][i]) {
          fail[nex[u][i]] = nex[fail[u]][i];
```

```
          q.push(nex[u][i]);
        } else {
          nex[u][i] = nex[fail[u]][i];
        }
      }
    }
  }
  void query(char *s) {
    static int deg[NN];
    static queue<int> q;
    int len = strlen(s);
    for (int i = 0, p = 0; i < len; ++i) {
      p = nex[p][s[i]-beg];
      ++vis[p];
      // for (int t = p; t; t = fail[t]) ++vis[t];
    }
    for (int i = 1; i <= cnt; ++i) ++deg[fail[i]];
    for (int i = 1; i <= cnt; ++i) if (!deg[i]) q.push(i);
    while (q.size()) {
      int u = q.front();
      q.pop();
      vis[fail[u]] += vis[u];
      if (--deg[fail[u]] == 0) q.push(fail[u]);
    }
  }
} ac;
```

37 后缀数组 |SA

$sa[i]$ 表示将所有后缀排序后第 i 小的后缀的编号
$rk[i]$ 表示后缀 i 的排名
性质:$sa[rk[i]] = rk[sa[i]] = i$
$lcp(i,j)$ 表示后缀 i 和后缀 j 的最长公共前缀 (的长度)
$height[i] = lcp(sa[i], sa[i-1])$
引理 $height[rk[i]] \geq height[rk[i-1]] - 1$
$lcp(sa[i], sa[j]) = \min\{height[i+1 \cdots j]\}$
不同子串数目:$\frac{n(n+1)}{2} - \sum_{i=2}^{n} height[i]$

37.1 $O(nlogn)$

```
int sa[N], rk[N<<1], height[N];
template <typename T> // s start from 1
inline void SA(const T *s, const int &n) {
#define cmp(x, y, w) oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y
    + w]
  static int oldrk[N<<1], id[N], px[N], cnt[N], m;
  // memset(oldrk+n+1, 0, sizeof(int)*n); // multi testcase
  memset(cnt, 0, sizeof(int) * (m = 128));
  for (int i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
  for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
  for (int i = n; i; --i) sa[cnt[rk[i]]--] = i;
  for (int w = 1, p, i; w <= n; w <<= 1, m = p) {
```

```
    for (p = 0, i = n; i > n - w; --i) id[++p] = i;
    for (i = 1; i <= n; ++i) if (sa[i] > w) id[++p] = sa[i] - w;
    memset(cnt + 1, 0, sizeof(int) * m);
    for (i = 1; i <= n; ++i) ++cnt[px[i] = rk[id[i]]];
    for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for (i = n; i; --i) sa[cnt[px[i]]--] = id[i];
    swap(oldrk, rk); // memcpy(oldrk+1, rk+1, sizeof(int)*n);
    for (p = 0, i = 1; i <= n; ++i) rk[sa[i]] = cmp(sa[i], sa[i - 1],
        w) ? p : ++p;
  }
  for (int i = 1, k = 0; i <= n; ++i) {
    if (k) --k;
    while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;
    height[rk[i]] = k;
  }
#undef cmp
}
```

37.2 $O(n)$

```
namespace SuffixArray {

int sa[N], rk[N], ht[N];
bool t[N << 1];

inline bool islms(const int i, const bool *t) { return i > 0 && t[i]
   && !t[i - 1]; }

template <class T>
inline void sort(T s, int *sa, const int len, const int sz, const int
    sigma, bool *t, int *b, int *cb, int *p) {
  memset(b, 0, sizeof(int) * sigma);
  memset(sa, -1, sizeof(int) * len);
  for (register int i = 0; i < len; i++) b[static_cast<int>(s[i])]++;
  cb[0] = b[0];
  for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i];
  for (register int i = sz - 1; i >= 0; i--) sa[--cb[static_cast<int
     >(s[p[i]])]] = p[i];
  for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i -
     1];
  for (register int i = 0; i < len; i++)
    if (sa[i] > 0 && !t[sa[i] - 1])
      sa[cb[static_cast<int>(s[sa[i] - 1])]++] = sa[i] - 1;
  cb[0] = b[0];
  for (register int i = 1; i < sigma; i++) cb[i] = cb[i - 1] + b[i];
  for (register int i = len - 1; i >= 0; i--)
    if (sa[i] > 0 && t[sa[i] - 1])
      sa[--cb[static_cast<int>(s[sa[i] - 1])]] = sa[i] - 1;
}

template <class T>
inline void sais(T s, int *sa, const int len, bool *t, int *b, int *
   b1, const int sigma) {
  register int i, j, x, p = -1, cnt = 0, sz = 0, *cb = b + sigma;
  for (t[len - 1] = 1, i = len - 2; i >= 0; i--) t[i] = s[i] < s[i +
     1] || (s[i] == s[i + 1] && t[i + 1]);
```

```
  for (i = 1; i < len; i++)
    if (t[i] && !t[i - 1])
      b1[sz++] = i;
  sort(s, sa, len, sz, sigma, t, b, cb, b1);
  for (i = sz = 0; i < len; i++)
    if (islms(sa[i], t))
      sa[sz++] = sa[i];
  for (i = sz; i < len; i++) sa[i] = -1;
  for (i = 0; i < sz; i++) {
    for (x = sa[i], j = 0; j < len; j++) {
      if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) {
        cnt++, p = x;
        break;
      } else if (j > 0 && (islms(x + j, t) || islms(p + j, t))) {
        break;
      }
    }
    sa[sz + (x >>= 1)] = cnt - 1;
  }
  for (i = j = len - 1; i >= sz; i--)
    if (sa[i] >= 0)
      sa[j--] = sa[i];
  register int *s1 = sa + len - sz, *b2 = b1 + sz;
  if (cnt < sz)
    sais(s1, sa, sz, t + len, b, b1 + sz, cnt);
  else
    for (i = 0; i < sz; i++) sa[s1[i]] = i;
  for (i = 0; i < sz; i++) b2[i] = b1[sa[i]];
  sort(s, sa, len, sz, sigma, t, b, cb, b2);
}

template <class T>
inline void getHeight(T s, int n) {
  for (register int i = 1; i <= n; i++) rk[sa[i]] = i;
  register int j = 0, k = 0;
  for (register int i = 0; i < n; ht[rk[i++]] = k)
    for (k ? k-- : 0, j = sa[rk[i] - 1]; s[i + k] == s[j + k]; k++)
      ;
}

template <class T>  // s start from 0
inline void init(T s, const int len, const int sigma = 128) {
  sais(s, sa, len + 1, t, rk, ht, sigma);
  getHeight(s, len);
  for (int i = 1; i <= len; ++i) ++sa[i];
  for (int i = len; i; --i) rk[i] = rk[i-1];
}

}  // namespace SuffixArray
```

37.3 树上 SA

树上可能出现完全相同的字符串，增加上一轮的有序状态 rk 为”第三关键字”

```
struct SAonTree {
  int n, d[N], cnt[N], sa[N], rk[N<<1], _rk[N<<1], _oldrk[N<<1], tp[N
    <<1];
```

```
  template <typename T>
  void tsort(int *sa, T *rk, int *tp, int m) {
    memset(cnt, 0, sizeof(int)*(m+1));
    for (int i = 1; i <= n; ++i) ++cnt[rk[i]];
    for (int i = 1; i <= m; ++i) cnt[i] += cnt[i-1];
    for (int i = n; i; --i) sa[cnt[rk[tp[i]]]--] = tp[i];
  }
  template <typename T>
  void build(int *f, const T *a, const int n) {
    this->n = n;
    int p = 128, i; // p = n
    iota(tp+1, tp+n+1, 1);
    tsort(sa, a, tp, p);
    for (i = 1, p = 0; i <= n; ++i) {
      _rk[sa[i]] = a[sa[i-1]] == a[sa[i]] ? p : ++p;
      rk[sa[i]] = i;
    }
    for (int w = 1, t = 0; w < n; w <<= 1, ++t) {
      for (i = 1; i <= n; ++i) _oldrk[i] = rk[f[i]];
      tsort(tp, _oldrk, sa, n);
      tsort(sa, _rk, tp, p);
      swap(_rk, tp);
      for (i = 1, p = 0; i <= n; ++i) {
        _rk[sa[i]] = tp[sa[i-1]] == tp[sa[i]]
            && tp[f[sa[i-1]]] == tp[f[sa[i]]] ? p : ++p;
        rk[sa[i]] = i;
      }
      for (int i = n; i; --i) f[i] = f[f[i]]; // attention special
         tree
    }
  }
};
```

38 后缀平衡树

39 后缀自动机 |SAM

一个状态表示一个 $endpos$ 的等价类
$len(v)$ 为该状态最长的字符串长度
后缀链接 $link(v)$ 连接到对应于该状态最长字符串的最长后缀的另一个 $endpos$ 等价类的状态。

39.1 代码

```
struct SAM { // root 0
  static const int A = 26;
  static const int M = N<<1;
  static const char C = 'a';
  int sz, last, len[M], link[M], nex[M][A];
  void init() {
    memset(nex, 0, sizeof(int)*A*sz);
    link[0] = -1; sz = 1; last = 0;
  }
```

```
  int extend(int c) {
    int cur = sz++, p = last;
    len[cur] = len[last]+1;
    for (; ~p && !nex[p][c]; p = link[p]) nex[p][c] = cur;
    if (p == -1) return link[cur] = 0, cur;
    int q = nex[p][c];
    if (len[p]+1 == len[q]) return link[cur] = q, cur;
    int clone = sz++;
    memcpy(nex[clone], nex[q], sizeof nex[q]);
    link[clone] = link[q];
    len[clone] = len[p]+1;
    for (; ~p && nex[p][c] == q; p = link[p]) nex[p][c] = clone;
    link[q] = link[cur] = clone;
    return cur;
  }
  void insert(const string &s) {
    init(); for (char ch : s) last = extend(ch-C);
  }
};
```

```
  void build() { // topo on parent tree
    static int t[M], rk[M];
    memset(t, 0, sizeof(int)*sz);
    for (int i = 0; i < sz; ++i) ++t[len[i]];
    for (int i = 1; i < sz; ++i) t[i] += t[i-1];
    for (int i = 0; i < sz; ++i) rk[--t[len[i]]] = i;
    for (int _ = sz-1, i, j; _; --_) { // assert(rk[0] == 0);
      i = rk[_];
      j = link[i];
      cnt[j] += cnt[i];
    }
  }
```

39.2 检查字符串是否出现

丢进去转移。这个算法还找到了模式串在文本串中出现的最大前缀长度。

39.3 不同子串个数

不同子串的个数等于自动机中以 t_0 为起点的不同路径的条数-1(空串)。令 d_v 为从状态 v 开始的路径数量（包括长度为零的路径）

$$d_v = 1 + \sum_{w:(v,w,c)\in DAWG} d_w$$

或利用上述后缀自动机树形结构统计节点对应的子串数量 $len(i) - len(link(i))$

ps: 若新增一个字符，其增量为 $len(cur) - len(link(cur))$ 不包括 $clone$ 结点

39.4 所有不同子串的总长度

$$ans_v = \sum_{w:(v,w,c)\in DAWG} d_w + ans_w$$

法二：每个节点对应的所有后缀长度是 $\frac{len(i)\times(len(i)+1)}{2}$，减去其 $link$ 节点的对应值就是该节点的净贡献

39.5 字典序第 k 大子串

不同位置的相同子串算作一个，每个非 clone 状态的数量记为 1
不同位置的相同子串算作多个，每个状态的数量为 parent 树上求子树和
在 SAM 的 DAG 求和然后字典序搞搞

39.6 两个字符串的最长公共子串

```
int lcs(const string &s) {
  int u = 0, l = 0, res = 0, c;
  for (char ch : s) {
    c = ch-C;
    while (u && !nex[u][c]) l = len[u = link[u]];
    if (nex[u][c]) u = nex[u][c], ++l;
    res = max(res, l);
  }
  return res;
}
```

39.7 多个字符串间的最长公共子串

记录 $f[i][j]$ 为第 i 个字符串在 sam 上状态 j 的匹配长度
$ans = \max\limits_{j}(\min\limits_{i} f[i][j])$

39.8 后缀的最长公共前缀 |height

求两个后缀的最长公共前缀，显然就是两个后缀的节点在 Parent 树上的 LCA

40 广义后缀自动机

广义后缀自动机 (General Suffix Automaton) 是将后缀自动机整合到字典树中来解决对于多个字符串的子串问题

40.1 离线构造

```
struct generalSAM {
  static const int A = 26;
  static const int M = N<<1;
  static const char C = 'a';
  int sz, len[M], link[M], nex[M][A];
  generalSAM() { init(); }
  void init() {
    // memset(nex, 0, sizeof(int)*A*sz);
    link[0] = -1; sz = 1;
  }
  int insertSAM(int last, int c) {
    int cur = nex[last][c];
    if (len[cur]) return cur;
    len[cur] = len[last]+1;
    int p = link[last];
    for (; ~p && !nex[p][c]; p = link[p]) nex[p][c] = cur;
```

```
    if (p == -1) return link[cur] = 0, cur;
    int q = nex[p][c];
    if (len[p]+1 == len[q]) return link[cur] = q, cur;
    int clone = sz++;
    for (int i = 0; i < A; ++i)
      nex[clone][i] = len[nex[q][i]] ? nex[q][i] : 0;
    len[clone] = len[p]+1;
    for (; ~p && nex[p][c] == q; p = link[p]) nex[p][c] = clone;
    link[clone] = link[q];
    link[q] = link[cur] = clone;
    return cur;
  }
  int insertTrie(int cur, int c) {
    return nex[cur][c] ? nex[cur][c] : nex[cur][c] = sz++;
  }
  void insert(const string &s) {
    int last = 0; for (char ch : s) last = insertTrie(last, ch-C);
  }
  void insert(const char *s, int n) {
    for (int i = 0, last = 0; i < n; ++i) last = insertTrie(last, s[i
       ]-C);
  }
  void build() {
    queue<pair<int, int>> q;
    for (int i = 0; i < A; ++i)
      if (nex[0][i]) q.push({0, i});
    while (!q.empty()) {
      auto item = q.front(); q.pop();
      int last = insertSAM(item.first, item.second);
      for (int i = 0; i < A; ++i)
        if (nex[last][i]) q.push({last, i});
    }
  }
};
```

40.2 在线构造

```
struct generalSAM {
  static const int A = 26;
  static const int M = N<<1;
  static const char C = 'a';
  int sz, len[M], link[M], nex[M][A];
  generalSAM() { init(); }
  void init() {
    memset(nex, 0, sizeof(int)*A*sz);
    link[0] = -1; sz = 1;
  }
  int extend(int last, int c) {
    if (nex[last][c]) {
      int p = last, cur = nex[p][c];
      if (len[p]+1 == len[cur]) return cur;
      int q = sz++;
      len[q] = len[p]+1;
      memcpy(nex[q], nex[cur], sizeof nex[q]);
      for ( ; ~p && nex[p][c] == cur; p = link[p]) nex[p][c] = q;
```

```
      link[q] = link[cur];
      link[cur] = q;
      return q;
    }
    int cur = sz++, p = last;
    len[cur] = len[p]+1;
    for ( ; ~p && !nex[p][c]; p = link[p]) nex[p][c] = cur;
    if (p == -1) return link[cur] = 0, cur;
    int q = nex[p][c];
    if (len[p]+1 == len[q]) return link[cur] = q, cur;
    int clone = sz++;
    len[clone] = len[p]+1;
    memcpy(nex[clone], nex[q], sizeof nex[q]);
    for (; ~p && nex[p][c] == q; p = link[p]) nex[p][c] = clone;
    link[clone] = link[q];
    link[q] = link[cur] = clone;
    return cur;
  }
  void insert(const string &s) {
    int last = 0; for (char ch : s) last = extend(last, ch-C);
  }
  void insert(const char *s) {
    for (int i = 0, last = 0; s[i]; ++i) last = extend(last, s[i]-C);
  }
};
```

40.3 多个字符串间的最长公共子串

设有 k 个字符串，每个结点建立长度为 k 的标记，在 parent 树自底向上合并，若满足所有标记，则记录

(对于本题而言，可以仅为标记数组，若需要求出此子串的个数，则需要改成计数数组)(可用二进制或 bitset)

40.4 根号暴力

在 parent 树上从字符串的每一个前缀的状态暴力往上跳 (须标记 vis)
例如可用此法记录上述的标记数组
证明: 对于第 i 个字符串，它最多会贡献 $\min(n, |s_i|^2), n = \sum |s_i|, O(n\sqrt{n})$

```
  void jump(const string &s, int id) {
    int x = 0;
    for (char ch : s) {
      x = nex[x][ch-C];
      for (int y = x; y && vis[y] != id; y = link[y]) {
        vis[y] = id;
        ++k[y]; // 记录信息
      }
    }
  }
```

40.5 树上本质不同路径数

一颗无根树上任意一条路径必定可以在以某个叶节点为根时，变成一条从上到下的路径
暴力把所有叶子结点为根的树加入自动机，就这?

```cpp
void dfs(int u, int fa = 0, int last = 0) {
  int nex = gsam.extend(last, a[u]);
  for (int v : e[u]) if (v != fa) dfs(v, u, nex);
}
```

40.6 循环同构问题

CF235C:SAM 读入字符串是支持删除首字符的! 将一个字符串所有的循环同构串插入到 sam, 外加 vis 去重

```cpp
ll cyclic_query(const string &s) { // 循环同构
    static int id = 0, vis[M];
    ++id;
    ll res = 0;
    for (int i = 0, c, m = s.size(), u = 0, l = 0; i < m*2-1; ++i) {
      c = s[i%m]-C;
      while (u && !nex[u][c]) l = len[u = link[u]];
      if (nex[u][c]) u = nex[u][c], ++l;
      if (l > m && --l == len[link[u]]) u = link[u]; // 删除首字符
      if (l == m && vis[u] != id) res += cnt[u], vis[u] = id; // 去重,记录答案
    }
    return res;
  }
```

41 最小表示法

$S[i\cdots n]+S[1\cdots i-1]=T$ 则称 S 与 T ** 循环同构 **
字符串 S 的 ** 最小表示 ** 为与 S 循环同构的所有字符串中字典序最小的字符串
$O(n)$

```cpp
int min_cyclic_string(const string &s) {
  int k = 0, i = 0, j = 1, n = s.size();
  while (k < n && i < n && j < n) {
    if (s[(i + k) % n] == s[(j + k) % n]) {
      k++;
    } else {
      s[(i + k) % n] > s[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
      if (i == j) i++;
      k = 0;
    }
  }
  return min(i, j);
}
```

42 Lyndon 分解

Lyndon 串：对于字符串 s，如果 s 的字典序严格小于 s 的所有后缀的字典序，我们称 s 是简单串，或者 Lyndon 串。

Lyndon 分解：串 s 的 Lyndon 分解记为 $s = w_1 w_2 \cdots w_k$，其中所有 w_i 为简单串，并且他们的字典序按照非严格单减排序，即 $w_1 \geq w_2 \geq \cdots \geq w_k$。可以发现，这样的分解存在且唯一。

Duval 可以在 $O(n)$ 的时间内求出一个串的 Lyndon 分解。

```
// duval_algorithm
vector<string> duval(string const& s) {
  int n = s.size(), i = 0;
  vector<string> factorization;
  while (i < n) {
    int j = i + 1, k = i;
    while (j < n && s[k] <= s[j]) {
      if (s[k] < s[j])
        k = i;
      else
        k++;
      j++;
    }
    while (i <= k) {
      factorization.push_back(s.substr(i, j - k));
      i += j - k;
    }
  }
  return factorization;
}
```

43 回文树 | 回文自动机 |PAM

定理：对于一个字符串 s，它的本质不同回文子串个数最多只有 $|s|$ 个。

状态数，复杂度 $O(n)$

由于回文树的构造过程中，节点本身就是按照拓扑序插入，因此可以按序枚举所有状态实现树遍历

```
struct PAM {
  static const int A = 26;
  static const char C = 'a';
  int sz, tot, last;
  int ch[N][A], len[N], fail[N];
  char s[N];
  PAM() { init(); }
  int node(int l) {
    ++sz;
    memset(ch[sz], 0, sizeof ch[sz]);
    len[sz] = l;
    fail[sz] = 0;
    return sz;
  }
  void init() {
    sz = -1;
    last = 0;
    s[tot = 0] = '#';
    node(0);
    fail[0] = node(-1);
  }
  int getfail(int x) {
    while (s[tot-len[x]-1] != s[tot]) x = fail[x];
```

```
    return x;
  }
  void insert(char c) {
    s[++tot] = c;
    int now = getfail(last);
    if (!ch[now][c-C]) {
      int x = node(len[now]+2);
      fail[x] = ch[getfail(fail[now])][c-C];
      ch[now][c-C] = x;
    }
    last = ch[now][c-C];
  }
};
```

43.1 特殊 log 性质

记字符串 s 长度为 i 的前缀为 $pre(s,i)$，长度为 i 的后缀为 $suf(s,i)$

周期：若 $0<p\le|s|$，$\forall 1\le i\le|s|-p, s[i]=s[i+p]$，就称 p 是 s 的周期。

border：若 $0\le r<|s|$，$pre(s,r)=suf(s,r)$，就称 $pre(s,r)$ 是 s 的 border。

周期和 border 的关系：t 是 s 的 border，当且仅当 $|s|-|t|$ 是 s 的周期。

引理 1：t 是回文串 s 的后缀，t 是 s 的 border 当且仅当 t 是回文串。

引理 2：t 是回文串 s 的 border ($|s|\le 2|t|$)，s 是回文串当且仅当 t 是回文串。

引理 3：t 是回文串 s 的 border，则 $|s|-|t|$ 是 s 的周期，$|s|-|t|$ 为 s 的最小周期，当且仅当 t 是 s 的最长回文真后缀。

引理 4：x 是一个回文串，y 是 x 的最长回文真后缀，z 是 y 的最长回文真后缀。令 u,v 分别为满足 $x=uy, y=vz$ 的字符串，则有下面三条性质

$|u|\ge|v|$;

如果 $|u|>|v|$，那么 $|u|>|z|$;

如果 $|u|=|v|$，那么 $u=v$。

推论：s 的所有回文后缀按照长度排序后，可以划分成 $\log|s|$ 段等差数列。

回文树上的每个节点 u 需要多维护两个信息，$diff[u]$ 和 $slink[u]$。$diff[u]$ 表示节点 u 和 $fail[u]$ 所代表的回文串的长度差，即 $len[u]-len[fail[u]]$。$slink[u]$ 表示 u 一直沿着 fail 向上跳到第一个节点 v，使得 $diff[v]\neq diff[u]$，也就是 u 所在等差数列中长度最小的那个节点。

```
      diff[x] = len[x]-len[fail[x]];
      slink[x] = diff[x] == diff[fail[x]] ? slink[fail[x]] : fail[x];
```

```
for (int i = 2; i <= sz; ++i)
  for (int j = i, k = slink[i]; j; j = k, k = slink[j])
    // 等差数列[len[k], len[j]] d = diff[j]
```

43.2 最小回文划分

问题描述：求最小的 k，使得字符串能分成 k 段且每段都是回文串

暴力:$dp[i]=1+\min\limits_{s[j+1\cdots i]} dp[j]$

$g[v]$ 表示 v 所在等差数列的 dp 值之和，且 v 是这个等差数列中长度最长的节点，则 $g[v]=\sum_{x,slink[x]=v} dp[i-len[x]]$，这里 i 是当前枚举到的下标。(该题改求和为取 min)

假设当前枚举到第 i 个字符,回文树上对应节点为 x。$g[x]=g[fail[x]]+dp[i-(len[slink[x]]+diff[x])]$

```
  void solve() {
    dp[0] = 1;
    for (int x = 0, i = 1; i <= tot; ++i) {
      while (s[i-len[x]-1] != s[i]) x = fail[x];
      x = ch[x][s[i]-C];
      for (int j = x; j > 1; j = slink[j]) {
        g[j] = dp[i-len[slink[j]]-diff[j]];
        if (diff[j] == diff[fail[j]]) update(g[j], g[fail[j]]);
        update(dp[i], g[j]);
      }
    }
  }
```

43.3 回文级数优化回文树上 dp

问题描述：把字符串 s 划分成若干个字符串 $t_1t_2\cdots t_k$，使得 $t_i = t_{k-i+1}$，求方案数

问题转化：将字符串转为 $s_1s_ns_2s_{n-1}\cdots$，原划分方案恰好对应了对新串进行偶数长度的回文划分的方案

tips：只需要考虑偶数下标位置的 dp 值即可

44 序列自动机

序列自动机是接受且仅接受一个字符串的子序列的自动机。

```
struct SeqAutomaton { // suppose string [1, n]
  static const int A = 26;
  static const char C = 'a';
  int nex[N][A];
  void build(const string &s) {
    memset(nex[s.size()], 0, sizeof nex[0]);
    for (int i = s.size(); i; --i) {
      memcpy(nex[i-1], nex[i], sizeof nex[0]);
      nex[i-1][s[i-1]-C] = i;
    }
  }
};
```

45 Main-Lorentz 算法

我们将一个字符串连续写两遍所产生的新字符串称为重串 (tandem repetition)。将被重复的这个字符串称为原串。

如果一个重串的原串不是重串，则我们称这个重串为本原重串 (primitive repetition)。可以证明，本原重串最多有 $O(n\log n)$ 个。

如果我们把一个重串用 Crochemore 三元组 (i,p,r) 进行压缩，其中 i 是重串的起始位置，p 是该重串某个循环节的长度（注意不是原串长度!），r 为这个循环节重复的次数。则某字符串的所有重串可以被 $O(n\log n)$ 个 Crochemore 三元组表示。

Part V
图论 | 树论

46 DFS 树

47 树的重心

```
void treedp(int cur, int fa) {
  s[cur] = c[cur];
  for(int i = fir[cur]; i; i = nex[i]) {
    if(e[i] == fa) continue;
    treedp(e[i], cur);
    s[cur] += s[e[i]];
    maxs[cur] = max(maxs[cur], s[e[i]]);
  }
  maxs[cur] = max(maxs[cur], sum-s[cur]);
}
```

48 最大团

最大独立集数 = 补图的最大团

```
struct MaxClique {
  vector<int> res, tmp, cnt;
  bool dfs(int p) {
    for (int i = p+1, flag; i <= n; ++i) {
      if (cnt[i]+tmp.size() <= res.size()) return false;
      if (!g[p][i]) continue;
      flag = 1;
      for (int j : tmp)
        if (!g[i][j]) flag = 0;
      if (!flag) continue;
      tmp.push_back(i);
      if (dfs(i)) return true;
      tmp.pop_back();
    }
    if (tmp.size() > res.size()) {
      res = tmp;
      return true;
    }
    return false;
  }
  void solve() {
    vector<int>(n+1, 0).swap(cnt);
    vector<int>().swap(res);
    for (int i = n; i; --i) {
      vector<int>(1, i).swap(tmp);
      dfs(i);
      cnt[i] = res.size();
    }
  }
} MC;
```

49 稳定婚姻匹配

```
template <typename T = int> struct Stable_Marriage {
  int t[N], b[N], g[N], rkb[N][N], rkg[N][N];
  T wb[N][N], wg[N][N];
  queue<int> q;
  void init(const int &n) {
    queue<int>().swap(q);
    memset(t, 0, sizeof(int)*(n+3));
    memset(b, 0, sizeof(int)*(n+3));
    memset(g, 0, sizeof(int)*(n+3));
    for (int i = 1; i <= n; ++i) {
      q.push(i);
      for (int j = 1; j <= n; ++j)
        rkb[i][j] = rkg[i][j] = j;
      sort(rkb[i]+1, rkb[i]+n+1,
         [&](const int &x, const int &y) { return wb[i][y] < wb[i][x
            ]; });
    }
  }
  bool match(const int &x, const int &y) {
    if (g[y]) {
      if (wg[y][x] < wg[y][g[y]]) return false;
      b[g[y]] = 0;
      q.push(g[y]);
    }
    b[x] = y; g[y] = x;
    return true;
  }
  void gale_shapely(const int &n) {
    init(n);
    while (q.size()) {
      int x = q.front(); q.pop();
      int y = rkb[x][++t[x]];
      if (!match(x, y)) q.push(x);
    }
  }
};
```

50 最小生成树

Prim

```
inline void prim() {
  fill(dis, dis+n+1, INF);
  dis[1] = 0;
  for(int t = 1; t <= n; ++t)
  {
    int mini = 0;
    for(int i = 1; i <= n; ++i)
      if(!vis[i] && dis[i] < dis[mini])
        mini = i;
    vis[mini] = 1;
    ans += dis[mini];
    for(int i = 1; i <= n; ++i)
```

```
13        if(!vis[i]) dis[i] = min(dis[i], calc(mini, i));
14    }
15  }
```

51 二分图

51.1 二分图最大匹配

增广路算法 Augmenting Path Algorithm $O(nm)$

```
bool check(int u) {
  for (int v : e[u]) {
    if (vis[v]) continue;
    vis[v] = 1;
    if (!co[v] || check(co[v])) {
      co[v] = u;
      return true;
    }
  }
  return false;
}

inline int solve() {
  int res = 0;
  memset(co, 0, sizeof co);
  for (int i = 1; i <= n; ++i) {
    memset(vis, 0, sizeof(int)*(n+3));
    res += check(i);
  }
  return res;
}
```

网络流 $O(\sqrt{n}m)$

51.2 二分图最大权匹配

Hungarian Algorithm (Kuhn-Munkres Algorithm) 匈牙利算法又称为 KM 算法，可以在 $O(n^3)$ 时间内求出二分图的最大权完美匹配。

51.3 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。

51.4 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

52 最近公共祖先 |LCA

52.1 倍增

```cpp
struct LCA {
  static const int NN = (int)log2(N)+3;
  int f[N][NN], d[N], lg2[N];
  LCA() { for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1; }
  template <typename TT>
  void build(const TT e[], const int &u = 1, const int &fa = 0) {
    d[u] = d[fa]+1;
    f[u][0] = fa;
    for (int i = 1; (1<<i) <= d[u]; ++i)
      f[u][i] = f[f[u][i-1]][i-1];
    for (auto v : e[u]) if (v != fa)
      build(e, v, u);
  }
  int get(int x, int y) {
    if (d[x] < d[y]) swap(x, y);
    while (d[x] > d[y])
      x = f[x][lg2[d[x]-d[y]]];
    if (x == y) return x;
    for (int i = lg2[d[x]]; i >= 0; --i)
      if(f[x][i] != f[y][i])
        x = f[x][i], y = f[y][i];
    return f[x][0];
  }
};
```

52.2 树剖

```cpp
struct HLD {
  int fa[N], d[N], num[N], son[N], tp[N];
  vector<int> *e;
  void build(vector<int> *e, const int &rt = 1) {
    this->e = e;
    fa[rt] = 0;
    dfs1(rt);
    dfs2(rt);
  }
  void dfs1(const int &u = 1) {
    d[u] = d[fa[u]]+1;
    num[u] = 1;
    son[u] = 0;
    for (const int &v : e[u]) if (v != fa[u]) {
      fa[v] = u;
      dfs1(v);
      num[u] += num[v];
      if (num[v] > num[son[u]]) son[u] = v;
    }
  }
  void dfs2(const int &u = 1) {
    tp[u] = son[fa[u]] == u ? tp[fa[u]] : u;
    if (son[u]) dfs2(son[u]);
```

```
    for (const int &v : e[u]) if (v != son[u] && v != fa[u])
      dfs2(v);
  }
  int lca(int x, int y) {
    while (tp[x] != tp[y]) {
      if (d[tp[x]] < d[tp[y]]) swap(x, y);
      x = fa[tp[x]];
    }
    return d[x] < d[y] ? x : y;
  }
};
```

52.3 欧拉序

dfs 时进入一个节点加入序列，回溯回来也加一次
lca 转变为区间深度最小的点

52.4 带权 LCA

```
template <typename T>
struct LCA {
  static const int NN = (int)log2(N)+3;
  int f[N][NN], d[N], lg2[N];
  T w[N][NN], init_val = 0;
  LCA() {
    for (int i = 2; i < N; ++i) lg2[i] = lg2[i>>1]+1;
    init();
  }
  // set sum or min or max, and don't forget to set init_val
  T update(const T &x, const T &y) { return x+y; }
  void init(const int &n = N-1) {
    fill(w[0], w[0]+(n+1)*NN, init_val);
  }
  template <typename TT>
  void build(const TT e[], const int &u = 1, const int &fa = 0) {
    d[u] = d[fa]+1;
    f[u][0] = fa;
    for (int i = 1; (1<<i) <= d[u]; ++i) {
      f[u][i] = f[f[u][i-1]][i-1];
      w[u][i] = update(w[u][i-1], w[f[u][i-1]][i-1]);
    }
    for (auto v : e[u]) if (v.first != fa) {
      w[v.first][0] = v.second;
      build(e, v.first, u);
    }
  }
  T get(int x, int y) {
    T res = init_val;
    if (d[x] < d[y]) swap(x, y);
    while (d[x] > d[y]) {
      res = update(res, w[x][lg2[d[x]-d[y]]]);
      x = f[x][lg2[d[x]-d[y]]];
    }
    if (x == y) return res;
    for (int i = lg2[d[x]]; i >= 0; --i)
```

```
      if(f[x][i] != f[y][i]) {
        res = update(res, w[x][i]);
        res = update(res, w[y][i]);
        x = f[x][i], y = f[y][i];
      }
    return update(res, update(w[x][0], w[y][0]));
  }
};
```

53 树上差分

```
template <typename T>
struct Tree {
  T val[N];
  void update_point(const int &x, const int &y, const T &k) {
    int _lca = lca(x, y);
    val[x] += k; val[y] += k;
    val[_lca] -= k; val[f[_lca][0]] -= k;
  }
  void update_edge(const int &x, const int &y, const T &k) {
    int _lca = lca(x, y);
    val[x] += k; val[y] += k; val[_lca] -= 2*k;
  }
  void dfs(const int &u = 1, const int &fa = 0) {
    for (int v : e[u]) if (v != fa) {
      dfs(v, u);
      val[u] += val[v];
    }
  }
};
```

54 树链剖分

```
template <typename T>
struct HLD {
  int dfn;
  int fa[N], d[N], num[N], son[N], id[N], tp[N];
  T init_val[N];
  vector<int> *e;
  SegmentTree ST;
  void build(vector<int> *e, T *a, int n, int rt = 1) {
    this->e = e;
    fa[rt] = dfn = 0;
    dfs1(rt);
    dfs2(rt);
    for (int i = 1; i <= n; ++i)
      init_val[id[i]] = a[i];
    ST.build(init_val, n);
  }
  void dfs1(const int &u = 1) {
    d[u] = d[fa[u]]+1;
    num[u] = 1;
```

```
    son[u] = 0;
    for (const int &v : e[u]) if (v != fa[u]) {
      fa[v] = u;
      dfs1(v);
      num[u] += num[v];
      if (num[v] > num[son[u]]) son[u] = v;
    }
  }
  void dfs2(const int &u = 1) {
    tp[u] = son[fa[u]] == u ? tp[fa[u]] : u;
    id[u] = ++dfn;
    if (son[u]) dfs2(son[u]);
    for (const int &v : e[u]) if (v != son[u] && v != fa[u])
      dfs2(v);
  }
  void add_sons(const int &x, const T &k) { ST.add(id[x], id[x]+num[x]-1, k); }
  void add(int x, int y, const T &k, const int &is_edge = 0) {
    while (tp[x] != tp[y]) {
      if (d[tp[x]] < d[tp[y]]) swap(x, y);
      ST.add(id[tp[x]], id[x], k);
      x = fa[tp[x]];
    }
    if (d[x] > d[y]) swap(x, y);
    ST.add(id[x], id[y], k);
    if (is_edge) ST.add(id[x], -k);
  }
  T query_sons(const int &x) { return ST.query(id[x], id[x]+num[x]-1); }
  T query(const int &x) { return ST.query(id[x]); }
  T query(int x, int y) {
    T res = 0;
    while (tp[x] != tp[y]) {
      if (d[tp[x]] < d[tp[y]]) swap(x, y);
      res += ST.query(id[tp[x]], id[x]);
      x = fa[tp[x]];
    }
    if (d[x] > d[y]) swap(x, y);
    return res+ST.query(id[x], id[y]);
  }
};
```

55 网络流

55.1 最大流

55.1.1 Dinic

普通情况下 $O(n^2m)$ 二分图中 $O(\sqrt{n}m)$

```
template <typename T>
struct Dinic {
  struct EDGE {
    int v, nex;
    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;
```

```
      }
    }
    return flow-rest;
  }
};
```

55.2 费用流

55.2.1 ZKW_SPFA

```
template <typename T>
struct ZKW_SPFA {
  struct Edge {
    int v, nex;
    T w, c; // edge wight and cost
    Edge(const int &_v, const int &_nex, const T &_w, const T &_c) \
    : v(_v), nex(_nex), w(_w), c(_c) {}
  };
  vector<Edge> e;
  int n, s, t;
  int fir[N], vis[N];
  T maxflow, mincost;
  T dis[N];
  ZKW_SPFA() { e.reserve(N<<4); }
  void init(const int &_n) {
    n = _n;
    maxflow = mincost = 0;
    e.clear();
    memset(fir, -1, sizeof(int)*(n+3));
  }
  void add_edge(const int &u, const int &v, const T &w = 1, const T &
     c = 0) {
    e.emplace_back(v, fir[u], w, c); fir[u] = e.size()-1;
    e.emplace_back(u, fir[v], 0, -c); fir[v] = e.size()-1;
  }
  pair<T, T> work(const int &_s, const int &_t) {
    s = _s; t = _t;
    while (spfa()) {
      vis[t] = 1;
      while (vis[t]) {
        memset(vis, 0, sizeof(int)*(n+3));
        maxflow += dfs(s, INF);
      }
    }
    return {maxflow, mincost};
  }
  private:
  bool spfa() {
    memset(dis, 0x3f, sizeof(T)*(n+3));
    memset(vis, 0, sizeof(int)*(n+3));
    deque<int> q;
    q.push_back(t);
    dis[t] = 0;
    vis[t] = 1;
    while (q.size()) {
      int u = q.front(); q.pop_front();
```

```
      for (int i = fir[u], v; ~i; i = e[i].nex) {
        v = e[i].v;
        if (!e[i^1].w || dis[v] <= dis[u]+e[i^1].c) continue;
        dis[v] = dis[u]+e[i^1].c;
        if (vis[v]) continue;
        vis[v] = 1;
        if (q.size() && dis[v] < dis[q.front()]) q.push_front(v);
        else q.push_back(v);
      }
      vis[u] = 0;
    }
    return dis[s] < INF;
  }
  T dfs(const int &u, const T &flow) {
    vis[u] = 1;
    if (u == t || flow <= 0) return flow;
    T res, used = 0;
    for (int i = fir[u], v; ~i; i = e[i].nex) {
      v = e[i].v;
      if (vis[v] || !e[i].w || dis[u] != dis[v]+e[i].c) continue;
      res = dfs(v, min(e[i].w, flow-used));
      if (!res) continue;
      mincost += res*e[i].c;
      e[i].w -= res;
      e[i^1].w += res;
      used += res;
      if (used == flow) break;
    }
    return used;
  }
};
```

55.3 上下界网络流

55.4 全局最小割 StoerWagner

```
template <typename T>
T sw(int n, T dis[N][N]) {
  int s, t;
  T res = INF;
  vector<int> dap(n+1, 0), ord(n+1, 0), vis;
  vector<T> w;
  function<T(int)> proc = & {
    vis = vector<int>(n+1, 0);
    w = vector<T>(n+1, 0);
    w[0] = -1;
    for (int i = 1; i <= n-x+1; ++i) {
      int mx = 0;
      for (int j = 1; j <= n; ++j) {
        if (!dap[j] && !vis[j] && w[j] > w[mx]) mx = j;
      }
      vis[mx] = 1;
      ord[i] = mx;
      for (int j = 1; j <= n; ++j) {
        if (!dap[j] && ! vis[j]) w[j] += dis[mx][j];
```

```
      }
    }
    s = ord[n-x];
    t = ord[n-x+1];
    return w[t];
  };
  for (int i = 1; i < n; ++i) {
    res = min(res, proc(i));
    dap[t] = 1;
    for (int j = 1; j <= n; ++j) {
      dis[s][j] += dis[t][j];
      dis[j][s] += dis[j][t];
    }
  }
  return res;
}
```

56 最短路

56.1 SPFA

```
inline void SPFA() {
  fill(dis+1, dis+n+1, INT_MAX);
  dis[S] = 0;
  head = tail = 0;
  q[++tail] = S;
  while(head < tail) {
    int cur = q[++head];
    for(int i = fir[cur], to, tmp; i; i = nex[i]) {
      to = ver[i];
      tmp = dis[cur]+w[i];
      if(tmp >= dis[to]) continue;
      dis[to] = tmp;
      q[++tail] = to;
    }
  }
}
```

57 负环

```
// 返回true有负环,返回false没负环
inline bool SPFA() {
  q[++tail] = 1;
  vis[1] = 1;
  cnt[1] = 1;
  dis[1] = 0;
  while(head < tail) {
    int cur = q[(++head)%Maxn];
    vis[cur] = 0;
    for(int i = fir[cur], to; i; i = nex[i]) {
      to = ver[i];
      if(dis[cur]+w[i] < dis[to]) {
```

```
        dis[to] = dis[cur]+w[i];
        if(!vis[to]) {
          q[(++tail)%Maxn] = to;
          vis[to] = 1;
          if(++cnt[to] > n) return true;
        }
      }
    }
  }
  return false;
}
```

58 割点

```
void tarjan(int cur, int fa) {
  dfn[cur] = low[cur] = ++_dfn;
  int child = 0;
  for(auto i : e[cur]) {
    if(!dfn[i]) {
      child++;
      tarjan(i, fa);
      low[cur] = min(low[cur], low[i]);
      if(cur != fa && low[i] >= dfn[cur]) flag[cur] = 1;
    }
    low[cur] = min(low[cur], dfn[i]);
  }
  if(cur == fa && child >= 2) flag[cur] = 1;
}
```

59 SCC 强连通分量 |Tarjan

59.1 递归版本

```
int _dfn, _col, _top;
int dfn[N], low[N], vis[N], col[N], sta[N];

void tarjan(const int &u) {
  dfn[u] = low[u] = ++_dfn;
  vis[u] = 1;
  sta[++_top] = u;
  for (int v : e[u]) {
    if (!dfn[v]) {
      tarjan(v);
      low[u] = min(low[u], low[v]);
    } else if (vis[v]) {
      low[u] = min(low[u], low[v]);
    }
  }
  if (dfn[u] == low[u]) {
    ++_col;
    do {
      col[sta[_top]] = _col;
```

```
      vis[sta[_top]] = 0;
    } while (sta[_top--] != u);
  }
}
```

60 缩点

```
void tarjan(int u) {
  dfn[u] = low[u] = ++_dfn;
  vis[u] = 1;
  sta[++top] = u;
  for (int v : e[u]) {
    if (!dfn[v]) {
      tarjan(v);
      low[u] = min(low[u], low[v]);
    } else if (vis[v]) {
      low[u] = min(low[u], low[v]);
    }
  }
  if (dfn[u] == low[u]) {
    w_col[++_col] = 0;
    do {
      col[sta[top]] = _col;
      vis[sta[top]] = 0;
      w_col[_col] += w[sta[top]];
    } while (sta[top--] != u);
  }
}

inline void suodian() {
  for (int i = 1; i <= n; ++i) {
    if (!dfn[i]) tarjan(i);
  }
  for (int i = 1; i <= n; ++i) {
    for (int j : e[i]) {
      if (col[i] == col[j]) continue;
      e_col[col[i]].push_back(col[j]);
    }
  }
}
```

61 2-SAT

61.1 SCC Tarjan

$O(n+m)$

```
struct TWO_SAT { // node start from 0
  int top, _dfn, _scc;
  int dfn[N<<1], low[N<<1], stk[N<<1], scc[N<<1], res[N];
  vector<int> e[N<<1];
  void init(const int &n) {
    top = 0;
```

```
    memset(dfn, 0, sizeof(int)*n*2);
    memset(low, 0, sizeof(int)*n*2);
    memset(scc, 0, sizeof(int)*n*2);
    for (int i = 0; i < n<<1; ++i) vector<int>().swap(e[i]);
  }
  // if u then v
  void add_edge(const int &u, const int &v) {
    e[u].emplace_back(v);
  }
  void add_edge(const int &u, const int &uv, const int &v, const int
     &vv) {
    e[u<<1^uv].emplace_back(v<<1^vv);
  }
  // pt i ==> i<<1 && i<<1|1 ==> 0 && 1
  inline bool work(const int &n) {
    for (int i = 0; i <= n<<1; ++i)
      if (!dfn[i]) tarjan(i);
    for (int i = 0; i < n; ++i) {
      if (scc[i<<1] == scc[i<<1|1]) return false;
      res[i] = scc[i<<1] > scc[i<<1|1];
    }
    return true;
  }
  void tarjan(const int &u) {
    dfn[u] = low[u] = ++_dfn;
    stk[++top] = u;
    for (int &v : e[u]) {
      if (!dfn[v]) {
        tarjan(v);
        low[u] = min(low[u], low[v]);
      } else if (!scc[v]) {
        low[u] = min(low[u], dfn[v]);
      }
    }
    if (dfn[u] == low[u]) {
      ++_scc;
      do {
        scc[stk[top]] = _scc;
      } while (stk[top--] != u);
    }
  }
};
```

61.2 DFS

$O(nm)$ 所求结果字典序最小

```
struct TWO_SAT {
  int n, cnt;
  int res[N], mem[N<<1], mark[N<<1];
  vector<int> e[N<<1];
  void init(const int &_n) {
    n = _n;
    memset(mark, 0, sizeof(int)*n*2);
    for (int i = 0; i < n<<1; ++i) vector<int>().swap(e[i]);
  }
```

```
  // if u then v
  void add_edge(const int &u, const int &v) {
    e[u].emplace_back(v);
  }
  // pt i ==> i<<1 && i<<1|1 ==> 0 && 1
  void add_edge(const int &u, const int &uv, const int &v, const int
     &vv) {
    e[u<<1|uv].emplace_back(v<<1|vv);
  }
  // tag 0 any 1 smallest
  bool work() {
    for (int i = 0; i < n; ++i) {
      if (mark[i<<1] || mark[i<<1|1]) continue;
      cnt = 0;
      if (!dfs(i<<1)) {
        while (cnt) mark[mem[cnt--]] = 0;
        if (!dfs(i<<1|1)) return false;
      }
    }
    for (int i = 0; i < n<<1; ++i) if (mark[i]) res[i>>1] = i&1;
    return true;
  }
  bool dfs(const int &u) {
    if (mark[u^1]) return false;
    if (mark[u]) return true;
    mark[mem[++cnt] = u] = 1;
    for (int v : e[u]) if (!dfs(v)) return false;
    return true;
  }
};
```

62 虚树

```
vector<int> ve[N];
void virtual_tree_clear(const int &u = 1) {
  for (const int &v : ve[u]) virtual_tree_clear(v);
  ve[u].clear();
}

// return the root of virtual tree
int virtual_tree_build(int vset[], const int &k) {
  static int stk[N], top;
  // id ==> dfn rank, d ==> depth
  int *id = hld.id, *d = hld.d;
  sort(vset+1, vset+k+1, & {
    return id[x] < id[y];
  });
  top = 0;
  int x, z;
  for (int i = 1; i <= k; ++i) {
    if (top && (z = hld.lca(vset[i], stk[top])) != stk[top]) {
      x = stk[top--];
      while (top && d[stk[top]] > d[z]) {
        ve[stk[top]].emplace_back(x);
        x = stk[top--];
```

```
      }
      ve[z].emplace_back(x);
      if (!top || stk[top] != z) stk[++top] = z;
    }
    stk[++top] = vset[i];
  }
  x = stk[top--];
  while (top) {
    ve[stk[top]].emplace_back(x);
    x = stk[top--];
  }
  // if (x != 1) ve[1].emplace_back(x); // force root at 1
  return x;
}
```

63 线段树优化建图

```
template <typename T>
struct SegmentTreeGarph {
  struct TreeNode {
    int l, r;
    int ls, rs;
  } tr[N*3];
  vector<pair<int, T>> *e;
  int tot, root[2];
  // op [down, 0] [up, 1]
  template <typename E>
  void build(const int &n, E *_e) {
    tot = n;
    e = _e;
    for (int i = 1; i <= n; ++i) tr[i].l = tr[i].r = i;
    build(1, n, root[0], 0);
    build(1, n, root[1], 1);
  }
  void build(const int &l, const int &r, int &i, const int &op) {
    if (l == r) return i = l, void();
    i = ++tot;
    tr[i].l = l; tr[i].r = r;
    int mid = (l+r)>>1;
    build(l, mid, tr[i].ls, op);
    build(mid+1, r, tr[i].rs, op);
    e[op ? tr[i].ls : i].emplace_back(op ? i : tr[i].ls, 0);
    e[op ? tr[i].rs : i].emplace_back(op ? i : tr[i].rs, 0);
  }
  void insert(const int &o, const int &l, const int &r, const T &w,
      const int &op) {
    if (l == r) e[op ? l : o].emplace_back(op ? o : l, w);
    else insert(o, l, r, w, op, root[op]);
  }
  void insert(const int &o, const int &l, const int &r, const T &w,
      const int &op, const int &i) {
    if (tr[i].l >= l && tr[i].r <= r) {
      e[op ? i : o].emplace_back(op ? o : i, w);
      return;
    }
```

```
    int mid = (tr[i].l+tr[i].r)>>1;
    if (l <= mid) insert(o, l, r, w, op, tr[i].ls);
    if (r >  mid) insert(o, l, r, w, op, tr[i].rs);
  }
};
```

64 矩阵树定理 |Kirchhoff

解决一张图的生成树个数计数问题 (详情见 oi-wiki)

65 斯坦纳树

给定连通图 G 中的 n 个点 m 条边与 k 个关键点，连接 k 个关键点，使得生成树的所有边的权值和最小。

我们使用状态压缩动态规划来求解。用 $f(i,S)$ 表示以 i 为根的一棵树，包含集合 S 中所有点的最小边权值和。

65.1 边权最小

- 首先对连通的子集进行转移，$f(i,S) \leftarrow \min(f(i,S), f(i,T)+f(i,S-T))$ 。
- 在当前的子集连通状态下进行边的松弛操作，$f(i,S) \leftarrow \min(f(i,S), f(j,S)+w(j,i))$

复杂度 $O(n \times 3^k + m \log m \times 2^k)$

```
int dp[N][1<<K];
vector<pair<int, int>> e[N]; // e[u] = {w, v}
priority_queue<pair<int, int>> q;

void dijkstra(int s) {
  while (q.size()) {
    int ud = -q.top().first;
    int u = q.top().second;
    q.pop();
    if (ud > dp[u][s]) continue;
    for (int i = 0, v, w; i < (int)e[u].size(); ++i) {
      w = e[u][i].first;
      v = e[u][i].second;
      if (dp[v][s] <= dp[u][s]+w) continue;
      dp[v][s] = dp[u][s]+w;
      q.push({-dp[v][s], v});
    }
  }
}

int steiner_tree(int *p) { // p[] is key point
  memset(dp, 0x3f, sizeof dp);
  for (int i = 0; i < k; ++i) dp[p[i]][1<<i] = 0;
  for (int s = 1; s < 1<<k; ++s) {
    for (int i = 1; i <= n; ++i) {
      for (int t = s&(s-1); t; t = s&(t-1))
        dp[i][s] = min(dp[i][s], dp[i][s^t]+dp[i][t]);
      if (dp[i][s] != INF) q.push({-dp[i][s], i});
    }
    dijkstra(s);
```

```
  }
  return dp[p[0]][(1<<k)-1];
}
```

65.2 点权最小

- $f(i,S) \leftarrow \min(f(i,S), f(i,T)+f(i,S-T)-a_i)$ 。由于此处合并时同一个点 a_i ，会被加两次，所以减去。
- $f(i,S) \leftarrow \min(f(i,S), f(j,S)+w(j,i))$ 。

65.3 路径输出

```
void dfs(int u, int s){
    if(!pre[u][s].second) return;
    // print
    if(pre[u][s].first == u) dfs(u, s^pre[u][s].second);
    dfs(pre[u][s].first, pre[u][s].second);
}
```

66 树上背包

时间复杂度 $O(n^2)$

```
void dfs(int u) {
  num[u] = 1;
  for (int &v : e[u]) {
    dfs(v);
    for (int i = min(m, num[u]+num[v]); j; --j) {
      for (int j = max(0, i-num[u]); j <= min(num[v], i); ++j) { // i
         -j >= num[u]
        dp[u][i] = max(dp[u][i], dp[u][i-j]+dp[v][j]);
      }
    }
    num[u] += num[v];
  }
}
```

67 仙人掌

两点之间最短路

```
namespace Cactus {

#define log(x) (31-__builtin_clz(x))
typedef long long ll;
typedef pair<int, ll> pil;
const int NN = (int)log2(N)+3;
int n, _dfn, cnt;
int f[N<<1][NN], dep[N<<1], dfn[N], from[N];
// od 为仙人掌上距离, dis 为圆方树上距离, cir 为环上边权和
ll od[N], dis[N<<1], cir[N];
```

```cpp
vector<pil> *e, ce[N<<1];

bool dfs(int u) {
  dfn[u] = ++_dfn;
  for (const pil &edge : e[u]) {
    int v = edge.first;
    if (v == f[u][0]) continue;
    ll w = edge.second;
    if (!dfn[v]) {
      f[v][0] = u;
      dep[v] = dep[u]+1;
      od[v] = od[u]+w;
      if (!dfs(v)) return false;
      if (!from[v]) ce[u].emplace_back(v, w);
    } else if (dfn[v] < dfn[u]) {
      cir[++cnt] = od[u]-od[v]+w;
      ce[v].emplace_back(n+cnt, 0);
      for (int x = u; x != v; x = f[x][0]) {
        if (from[x]) return false;
        from[x] = cnt;
        ll ww = od[x]-od[v];
        ce[n+cnt].emplace_back(x, min(ww, cir[cnt]-ww));
      }
    }
  }
  return true;
}
// 帶权倍增LCA
void build_lca(int u) {
  for (int i = 1; (1<<i) <= dep[u]; ++i)
    f[u][i] = f[f[u][i-1]][i-1];
  for (const pil &edge : ce[u]) {
    int v = edge.first;
    dep[v] = dep[u]+1;
    dis[v] = dis[u]+edge.second;
    f[v][0] = u;
    build_lca(v);
  }
}

bool build(int _n, vector<pil> *_e) {
  n = _n; e = _e;
  _dfn = cnt = 0;
  if (!dfs(1)) return false;
  dep[1] = 1;
  build_lca(1);
  return true;
}

ll query(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  int x = u, y = v;
  while (dep[x] > dep[y]) x = f[x][log(dep[x]-dep[y])];
  if (x == y) return dis[u]-dis[v];
  for (int i = log(dep[x]); i >= 0; --i)
    if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];
  if (f[x][0] <= n) return dis[u]+dis[v]-2*dis[f[x][0]];
```

```
  ll d = abs(od[x]-od[y]);
  return dis[u]-dis[x]+dis[v]-dis[y]+min(d, cir[f[x][0]-n]-d);
}

} /* namespace Cactus */
```

67.1 仙人掌 DP

```
// from[i] 节点i与父亲的边所在环的编号
// tp[i]环i深度最小的节点编号, bm[i]环i深度最大的节点编号
int _dfn, cnt, fa[N], dfn[N], from[N], tp[N], bm[N];

bool dfs(int u) {
  dfn[u] = ++_dfn;
  for (const int &v : e[u]) {
    if (v == fa[u]) continue;
    if (!dfn[v]) {
      fa[v] = u;
      dfs(v);
      // dp
    } else if (dfn[v] < dfn[u]) {
      ++cnt;
      tp[cnt] = v; bm[cnt] = u;
      for (int x = u; x != v; x = fa[x]) {
        if (from[x]) return false;
        from[x] = cnt;
      }
    }
  }
}
```

68 补图 DFS

```
void bfs(int S = 1) {
  static set<int> st[2]; // 存储未访问的点
  static queue<int> q;
  memset(dis+1, -1, sizeof(int)*n);
  dis[S] = 0;
  q.push(S);
  st[0].clear(); st[1].clear();
  for (int i = 1; i <= n; ++i) if (i != S) st[0].insert(i);
  while (q.size()) {
    int u = q.front();
    q.pop();
    for (int v : e[u]) if (st[0].count(v)) {
      st[0].erase(v);
      st[1].insert(v);
    }
    for (auto v : st[0]) if (dis[v] == -1) {
      dis[v] = dis[u]+1;
      q.push(v);
    }
```

```
    swap(st[0], st[1]);
    st[1].clear();
  }
}
```

69 浅谈图模型上的随机游走问题

69.1 网格图

$$f(i)=\begin{cases}p_1f(i-1,j)+p_2f(i,j-1)+p_3f(i+1,j)+p_4f(i,j+1)+1, i^2+j^2\le R\\ 0, i^2+j^2<R\end{cases}$$

69.1.1 高斯消元 $O(R^6)$

69.1.2 直接消元法 $O(R^4)$

69.1.3 主元法 $O(R^3)$

70 树分治

70.1 点分治

按树的重心分治

```
namespace DFZ {
int vis[N], siz[N], mxs[N], cnt[K];
vector<int> pset;

void calc_size(int u, int fa = 0) { // 找重心
  siz[u] = 1; mxs[u] = 0;
  pset.emplace_back(u);
  for (auto p : e[u]) {
    int v = p.first;
    if (v == fa || vis[v]) continue;
    calc_size(v, u);
    siz[u] += siz[v];
    mxs[u] = max(mxs[u], siz[v]);
  }
}

void query(int u, int fa, int dis) {
  for (int i = 1; i <= m; ++i) if (q[i] >= dis) ans[i] |= cnt[q[i]-
     dis];
  for (auto p : e[u]) {
    int v = p.first;
    if (v == fa || vis[v]) continue;
    query(v, u, dis+p.second);
  }
}

void update(int u, int fa, int dis, int k) {
  if (dis < K) cnt[dis] += k;
```

```
  for (auto p : e[u]) {
    int v = p.first;
    if (v == fa || vis[v]) continue;
    update(v, u, dis+p.second, k);
  }
}

void dfz(int u = 1) {
  pset.clear();
  calc_size(u);
  for (int v : pset) { // get centre
    mxs[v] = max(mxs[v], (int)pset.size()-siz[v]);
    if (mxs[v] < mxs[u]) u = v;
  }
  cnt[0] = 1;
  for (auto p : e[u]) {
    int v = p.first;
    if (vis[v]) continue;
    query(v, u, p.second);
    update(v, u, p.second, 1);
  }
  update(u, 0, 0, -1); // clear
  vis[u] = 1;
  for (auto p : e[u]) {
    int v = p.first;
    if (vis[v]) continue;
    dfz(v);
  }
}
} // namespace 点分治
```

70.2 边分治

Part VI
数论

71 快排

```
void quick_sort(int l, int r) {
  if(l >= r) return;
  swap(a[l], a[l+rand()%(r-l)]);
  int i = l, j = r, mid = a[l];
  while(i < j) {
    while(i < j && a[j] >= mid) --j;
    swap(a[i], a[j]);
    while(i < j && a[i] < mid) ++i;
    swap(a[i], a[j]);
  }
  quick_sort(l, i-1);
  quick_sort(i+1, r);
}
```

72 求第 K 大数

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

73 求逆序对 (归并排序)

```
void merge_sort(int l, int r) {
  if(l == r) return;
  int mid = (l+r)>>1;
  merge_sort(l, mid);
  merge_sort(mid+1, r);
  int i = l, j = mid+1, k = l;
  while(k <= r) {
    if(j <= r && (i > mid || a[j] < a[i])) {
      ans += mid-i+1;
      b[k++] = a[j++];
    }
    else b[k++] = a[i++];
  }
  memcpy(a+l, b+l, sizeof(int)*(r-l+1));
}
```

74 线性基

求最大值 Luogu3812
求第 k 大数 HDOJ3949

```
template <typename T>
struct LinearBase {
  int sz = sizeof(T)*8, zero;
  T tot;
  vector<T> b, rb, p;
  LinearBase(){ init(); }
  void init() {
    tot = zero = 0;;
    vector<T>(sz, 0).swap(b);
    vector<T>().swap(rb);
    vector<T>().swap(p);
  }
  template <typename TT>
  void build(TT a[], const int &n) {
    init();
    for (int i = 1; i <= n; ++i) insert(a[i]);
  }
  void merge(const LinearBase xj) {
    for (int i : xj.b) if (i) insert(i);
  }
  void insert(T x) {
    for (int i = sz-1; i >= 0; --i) if ((x>>i)&1) {
      if (!b[i]) { b[i] = x; return; }
      x ^= b[i];
    }
```

```
    zero = 1;
  }
  bool find(T x) {
    for (int i = sz-1; i >= 0; --i) if ((x>>i)&1) {
      if (!b[i]) { return false; }
      x ^= b[i];
    }
    return true;
  }
  T max_xor() {
    T res = 0;
    for (int i = sz-1; i >= 0; --i)
      if (~(res>>i)&1) res ^= b[i];
      // res = max(res, res^b[i]);
    return res;
  }
  T min_xor() {
    if (zero) return 0;
    for (int i = 0; i < sz; ++i)
      if (b[i]) return b[i];
  }
  void rebuild() {
    rb = b;
    vector<T>().swap(p);
    for (int i = sz-1; i >= 0; --i)
      for (int j = i-1; j >= 0; --j)
        if ((rb[i]>>j)&1) rb[i] ^= rb[j];
    for (int i = 0; i < sz; ++i)
      if (rb[i]) p.emplace_back(rb[i]);
    tot = ((T)1<<p.size())+zero;
  }
  T kth_min(T k) {
    if (k >= tot || k < 1) return -1;
    if (zero && k == 1) return 0;
    if (zero) --k;
    T res = 0;
    for (int i = (int)p.size()-1; i >= 0; --i)
      if ((k>>i)&1) res ^= p[i];
    return res;
  }
  T kth_max(const T &k) {
    return kth_min(tot-k);
  }
};
```

前缀和线性基 vector 跑贼鸡儿慢

```
template <class T>
struct PreSumLB {
  int tot, sz = sizeof(T)*8;
  vector<T> b[N];
  vector<int> p[N];
  PreSumLB() { init(); }
  void init() {
    tot = 0;
    vector<T>(sz, 0).swap(b[0]);
    vector<int>(sz, 0).swap(p[0]);
  }
```

```
  void append(T val) {
    int pos = ++tot;
    vector<T> &bb = b[tot];
    vector<int> &pp = p[tot];
    pp = p[tot-1];
    bb = b[tot-1];
    for (int i = sz-1; i >= 0; --i) if ((val>>i)&1) {
      if (bb[i]) {
        if (pos > pp[i]) swap(pos, pp[i]), swap(val, bb[i]);
        val ^= bb[i];
      } else {
        bb[i] = val;
        pp[i] = pos;
        return;
      }
    }
  }
  T query(const int &l, const int &r) {
    T res = 0;
    vector<T> &bb = b[r];
    vector<int> &pp = p[r];
    for (int i = sz-1; i >= 0; --i)
      if (pp[i] >= l) res = max(res, res^bb[i]);
    return res;
  }
};
```

75 矩阵

75.1 矩阵快速幂

75.2 矩阵求逆

```
template <typename T>
struct Martix {
  int n, m;
  T a[N][N];
  Martix(){}
  Martix(const int &_n) : n(_n), m(_n) { init(); }
  Martix(const int &_n, const int &_m) : n(_n), m(_m) { init(); }
  T* operator [] (const int &i) { return a[i]; }
  void init(const int &tag = 0) {
    for (int i = 1; i <= n; ++i) memset(a[i], 0, sizeof(T)*(m+1));
    if (tag) for (int i = 1; i <= n; ++i) a[i][i] = tag;
  }
  friend Martix operator * (const Martix &m1, const Martix &m2) {
    Martix res(m1.n, m2.m);
    for (int i = 1; i <= res.n; ++i)
      for (int j = 1; j <= res.m; ++j)
        for (int k = 1; k <= m1.m; ++k)
          res.a[i][j] = (res.a[i][j]+m1.a[i][k]*m2.a[k][j])%MOD;
    return res;
  }
  Martix& operator *= (const Martix &mx) { return *this = *this*mx; }
```

```
22  Martix& operator + (const Martix &mx) const { Martix res(n, m);
       return res += mx; }
23  Martix& operator += (const Martix &mx) {
24    assert(n == mx.n && m == mx.m);
25    for (int i = 1; i <= n; ++i)
26      for (int j = 1; j <= m; ++j)
27        a[i][j] += mx.a[i][j];
28    return *this;
29  }
30  Martix& operator - (const Martix &mx) const { Martix res(n, m);
       return res -= mx; }
31  Martix& operator -= (const Martix &mx) {
32    assert(n == mx.n && m == mx.m);
33    for (int i = 1; i <= n; ++i)
34      for (int j = 1; j <= m; ++j)
35        a[i][j] -= mx.a[i][j];
36    return *this;
37  }
38  template <typename TT>
39  Martix pow(const TT &p) const {
40    Martix res(n, m), a = *this;
41    res.init(1);
42    for (TT i = p; i; i >>= 1, a *= a) if (i&1) res *= a;
43    return res;
44  }
45  Martix inv() const {
46    Martix res = *this;
47    vector<int> is(n+1), js(n+1);
48    for (int k = 1; k <= n; ++k) {
49      for (int i = k; i <= n; ++i)
50        for (int j = k; j <= n; ++j) if (res.a[i][j]) {
51          is[k] = i; js[k] = j; break;
52        }
53      for (int i = 1; i <= n; ++i) swap(res.a[k][i], res.a[is[k]][i])
          ;
54      for (int i = 1; i <= n; ++i) swap(res.a[i][k], res.a[i][js[k]])
          ;
55      if (!res.a[k][k]) return Martix(0);
56      res.a[k][k] = mul_inverse(res.a[k][k]); // get inv of number
57      for (int j = 1; j <= n; ++j) if (j != k)
58        res.a[k][j] = res.a[k][j]*res.a[k][k]%MOD;
59      for (int i = 1; i <= n; ++i) if (i != k)
60        for (int j = 1; j <= n; ++j) if (j != k)
61          res.a[i][j] = (res.a[i][j]+MOD-res.a[i][k]*res.a[k][j]%MOD)
              %MOD;
62      for (int i = 1; i <= n; ++i) if (i != k)
63        res.a[i][k] = (MOD-res.a[i][k]*res.a[k][k]%MOD)%MOD;
64    }
65    for (int k = n; k; --k) {
66      for (int i = 1; i <= n; ++i) swap(res.a[js[k]][i], res.a[k][i])
          ;
67      for (int i = 1; i <= n; ++i) swap(res.a[i][is[k]], res.a[i][k])
          ;
68    }
69    return res;
70  }
71  T det() {
```

```
    long long res = 1;
    Martix cpy = *this;
    for (int i = 1; i <= n; ++i) {
      for (int j = i+1; j <= n; ++j) while (cpy.a[j][i]) {
        long long t = cpy.a[i][i]/cpy.a[j][i];
        for (int k = i; k <= n; ++k)
          cpy.a[i][k] = (cpy.a[i][k]+MOD-t*cpy.a[j][k]%MOD)%MOD;
        swap(cpy.a[i], cpy.a[j]);
        res = -res;
      }
      res = res*cpy.a[i][i]%MOD;
    }
    return (res+MOD)%MOD;
  }
  friend ostream& operator << (ostream &os, const Martix<T> &mx) {
    for (int i = 1; i <= mx.n; ++i)
      for (int j = 1; j <= mx.m; ++j)
        os << mx.a[i][j] << " \n"[j==mx.m];
    return os;
  }
};
```

75.3 伍德伯里矩阵恒等式 |Woodbury matrix identity

解决矩阵修改求逆问题 hdoj6994
$(A+UCV)^{-1}=A^{-1}-A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}$
$A\in R^{n\times n},U\in R^{n\times k},C\in R^{k\times k},V\in R^{k\times n}$
矩阵求逆 $O(n^3)$，单次修改 $O(n^2)$
例如给矩阵 A 的第 i 行第 j 列增加 Δ ，若 $A_{3*3},(i,j)=(2,3)$

$$UCV=\begin{bmatrix}0&0&0\\0&0&\Delta\\0&0&0\end{bmatrix}$$

则有
$U_{n\times 1}=[0\quad 1\quad 0]^T,U_i=1$
$C_{1\times 1}=[\Delta]$
$V_{1\times n}=[0\quad 0\quad 1],V_j=1$
注意运算顺序 $(A^{-1}U)(C^{-1}+VA^{-1}U)^{-1}(VA^{-1})$

76 高斯消元

```
struct GaussElimination {
  double a[N][N];
  void init() { memset(a, 0, sizeof a); }
  void init(const int &n) {
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n+1; ++j)
        a[i][j] = 0;
  }
  // ans is a[i][n+1]
  bool solve(const int &n) {
```

```
    for (int i = 1, j, k; i <= n; ++i) {
      for (j = i+1, k = i; j <= n; ++j)
        if (abs(a[j][i]) > abs(a[k][i])) k = j;
      if (abs(a[k][i]) < eps) return false;
      swap(a[k], a[i]);
      for (j = 1; j <= n; ++j) if (i != j) {
        double d = a[j][i]/a[i][i];
        for (k = i+1; k <= n+1; ++k)
          a[j][k] -= d*a[i][k];
      }
    }
    for (int i = 1; i <= n; ++i) a[i][n+1] /= a[i][i];
    return true;
  }
};
```

76.1 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 : no solution, 0 : multi , 1 : one
template <typename T>
int XorGauss(T a[N], const int &n) {
  for (int i = 1, j, k; i <= n; ++i) {
    for (k = i; !a[k][i] && k <= n; ++k) {}
    if (k <= n) swap(a[k], a[i]);
    for (j = 1; j <= n; ++j) if (i != j && a[j][i])
      for (k = i; k <= n+1; ++k) a[j][k] ^= a[i][k];
      // a[j] ^= a[i]; // bitset<N> a[N]
  }
  for (int i = 1; i <= n; ++i) if (!a[i][i] && a[i][n+1]) return -1;
  for (int i = 1; i <= n; ++i) if (!a[i][i]) return 0;
  return 1;
}
// dfs(n, 0)
void dfs(const int &u, const int &num) {
  if (num >= res) return;
  if (u <= 0) { res = num; return; }
  if (a[u][u]) {
    int t = a[u][n+1];
    for (int i = u+1; i <= n; ++i) {
      if (a[u][i]) t ^= used[i];
    }
    dfs(u-1, num+t);
  } else { // 自由元
    dfs(u-1, num);
    used[u] = 1;
    dfs(u-1, num+1);
    used[u] = 0;
  }
}
```

77 拉格朗日插值

```cpp
template <typename T, typename H, typename P>
long long Largrange(const T &k, const int &n, const H x[], const P y[]) {
  k %= MOD;
  long long res = 0, s1 = 1, s2 = 1;
  for (int i = 1; i <= n; ++i, s1 = s2 = 1) {
    for (int j = 1; j <= n; ++j) if (i != j) {
      s1 = s1*(x[i]-x[j]+MOD)%MOD;
      s2 = s2*(k-x[j]+MOD)%MOD;
    }
    res = (res+y[i]*s2%MOD*mul_inverse(s1)%MOD)%MOD;
  }
  return res;
}
```

```cpp
template <typename T, typename P> // x[i] = i -> y[i] = f(i)
long long Largrange(const T &k, const int &n, const P y[]) {
  if (k <= n) return y[k];
  static long long pre[N], suf[N];
  long long res = 0;
  k %= MOD;
  pre[0] = suf[n+1] = 1;
  for (int i = 1; i <= n; ++i) pre[i] = pre[i-1]*(k-i)%MOD;
  for (int i = n; i >= 1; --i) suf[i] = suf[i+1]*(k-i)%MOD;
  for (int i = 1; i <= n; ++i) {
    res = (res+y[i]*(pre[i-1]*suf[i+1]%MOD)%MOD
      *mul_inverse(((n-i)&1 ? -1 : 1)*fac[i-1]*fac[n-i]%MOD)%MOD)%MOD;
  }
  return (res+MOD)%MOD;
}
```

78 快速幂

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

79 快速乘

```cpp
inline long long qmul(long long x, long long y, long long mo) {
  // x*y - floor(x*y/mo)*mo
  typedef unsigned long long ull;
  typedef long double ld;
  return ((ull)x*y-(ull)((ld)x/mo*y)*mo+mo)%mo;
```

```
}
```

80 复数

```
struct comp {
  typedef double T; // maybe long double ?
  T real, imag;
  comp (const double &_real = 0, const double &_imag = 0) : real(
     _real), imag(_imag) {}
  friend comp operator + (const comp &c1, const comp &c2) { return
     comp(c1.real+c2.real, c1.imag+c2.imag); }
  friend comp operator - (const comp &c1, const comp &c2) { return
     comp(c1.real-c2.real, c1.imag-c2.imag); }
  friend comp operator * (const comp &c1, const comp &c2) { return
     comp(c1.real*c2.real-c1.imag*c2.imag, c1.real*c2.imag+c1.imag*c2
     .real); }
  comp& operator += (const comp &c) { return *this = *this+c; }
  comp& operator -= (const comp &c) { return *this = *this-c; }
  comp& operator *= (const comp &c) { return *this = *this*c; }
  friend istream& operator >> (istream &is, comp &c) { return is >> c
     .real >> c.imag; }
  friend ostream& operator << (ostream &os, comp &c) { return os << c
     .real << setiosflags(ios::showpos) << c.imag << "i";}
  comp conjugate() { return comp(real, -imag); }
  friend comp conjugate(const comp &c) { return comp(c.real, -c.imag)
     ; }
};
```

81 快速傅里叶变换 |FFT

```
namespace FFT { // array [0, n)
const int SIZE = (1<<18)+3;
int len, bit;
int rev[SIZE];
// #define comp complex<long double>
void fft(comp a[], int flag = 1) {
  for (int i = 0; i < len; ++i)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int base = 1; base < len; base <<= 1) {
    comp w, wn = {cos(PI/base), flag*sin(PI/base)};
    for (int i = 0; i < len; i += base*2) {
      w = { 1.0, 0.0 };
      for (int j = 0; j < base; ++j) {
        comp x = a[i+j], y = w*a[i+j+base];
        a[i+j] = x+y;
        a[i+j+base] = x-y;
        w *= wn;
      }
    }
  }
}
void work(comp f[], const int &n, comp g[], const int &m) {
```

```
  len = 1; bit = 0;
  while (len < n+m) len <<= 1, ++bit;
  // multi-testcase
  for (int i = n; i < len; ++i) f[i] = 0;
  for (int i = m; i < len; ++i) g[i] = 0;
  for (int i = 0; i < len; ++i)
    rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
  fft(f, 1); fft(g, 1);
  for (int i = 0; i < len; ++i) f[i] *= g[i];
  fft(f, -1);
  for (int i = 0; i < n+m; ++i) f[i].real /= len;
}
} // namespace FFT
```

82 快速数论变换 |NTT

```
namespace NTT { // array [0, n)
const int SIZE = (1<<18)+3;
const int G = 3;
int len, bit;
int rev[SIZE];
long long f[SIZE], g[SIZE];
template <class T>
void ntt(T a[], int flag = 1) {
  for (int i = 0; i < len; ++i)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int base = 1; base < len; base <<= 1) {
    long long wn = qpow(G, (MOD-1)/(base*2)), w;
    if (flag == -1) wn = qpow(wn, MOD-2);
    for (int i = 0; i < len; i += base*2) {
      w = 1;
      for (int j = 0; j < base; ++j) {
        long long x = a[i+j], y = w*a[i+j+base]%MOD;
        a[i+j] = (x+y)%MOD;
        a[i+j+base] = (x-y+MOD)%MOD;
        w = w*wn%MOD;
      }
    }
  }
}
template <class T>
void work(T a[], const int &n, T b[], const int &m) {
  len = 1; bit = 0;
  while (len < n+m) len <<= 1, ++bit;
  for (int i = 0; i < n; ++i) f[i] = a[i];
  for (int i = n; i < len; ++i) f[i] = 0;
  for (int i = 0; i < m; ++i) g[i] = b[i];
  for (int i = m; i < len; ++i) g[i] = 0;
  for (int i = 0; i < len; ++i)
    rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
  ntt(f, 1); ntt(g, 1);
  for (int i = 0; i < len; ++i) f[i] = f[i]*g[i]%MOD;
  ntt(f, -1);
  long long inv = qpow(len, MOD-2);
```

```
  for (int i = 0; i < n+m-1; ++i) f[i] = f[i]*inv%MOD;
}
} // namespace NTT
```

83 任意模数 NTT|MTT

```
namespace MTT {
const int SIZE = (1<<18)+7;
const int Mod = MOD;
comp w[SIZE];
int bitrev[SIZE];
long long f[SIZE];
void fft(comp *a, const int &n) {
  for (int i = 0; i < n; ++i) if (i < bitrev[i]) swap(a[i], a[bitrev[i]]);
  for (int i = 2, lyc = n >> 1; i <= n; i <<= 1, lyc >>= 1)
    for (int j = 0; j < n; j += i) {
      comp *l = a + j, *r = a + j + (i >> 1), *p = w;
      for (int k = 0; k < i>>1; ++k) {
        comp tmp = *r * *p;
        *r = *l - tmp, *l = *l + tmp;
        ++l, ++r, p += lyc;
      }
    }
}
template <class T>
inline void work(T *x, const int &n, T *y, const int &m) {
  static int bit, L;
  static comp a[SIZE], b[SIZE];
  static comp dfta[SIZE], dftb[SIZE];

  for (L = 1, bit = 0; L < n+m-1; ++bit, L <<= 1);
  for (int i = 0; i < L; ++i) bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (bit - 1));
  for (int i = 0; i < L; ++i) w[i] = comp(cos(2 * PI * i / L), sin(2 * PI * i / L));

  for (int i = 0; i < n; ++i) (x[i] += Mod) %= Mod, a[i] = comp(x[i] & 32767, x[i] >> 15);
  for (int i = n; i < L; ++i) a[i] = 0;
  for (int i = 0; i < m; ++i) (y[i] += Mod) %= Mod, b[i] = comp(y[i] & 32767, y[i] >> 15);
  for (int i = m; i < L; ++i) b[i] = 0;
  fft(a, L), fft(b, L);
  for (int i = 0; i < L; ++i) {
    int j = (L - i) & (L - 1);
    static comp da, db, dc, dd;
    da = (a[i] + conjugate(a[j])) * comp(.5, 0);
    db = (a[i] - conjugate(a[j])) * comp(0, -.5);
    dc = (b[i] + conjugate(b[j])) * comp(.5, 0);
    dd = (b[i] - conjugate(b[j])) * comp(0, -.5);
    dfta[j] = da*dc + da*dd*comp(0, 1);
    dftb[j] = db*dc + db*dd*comp(0, 1);
  }
  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  } // namespace MTT
```

84 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);
```

85 快速沃尔什变换 |FWT

推导详解
公式参考
洛谷例题
复杂度 $O(n\log n)|O(n2^n)$
$FWT(A\pm B)=FWT(A)\pm FWT(B)$
$FWT(cA)=cFWT(A)$
定义 $\bigoplus$ 为任意集合运算
$FWT(A\bigoplus B)=FWT(A)\times FWT(B)$
求 $C_i=\sum\limits_{i=j\bigoplus k}a_jb_k$

85.1 或运算

$FWT(A)[i]=\sum\limits_{j|i=i}A[j]$
$FWT(A)=[FWT(A_0),FWT(A_0+A_1)]$
$IFWT(A)=[IFWT(A_0),IFWT(A_1)-IFWT(A_0)]$

85.2 与运算

$FWT(A)[i] = \sum_{i\&j=j} A[j]$
$FWT(A) = [FWT(A_0 + A_1), FWT(A_1)]$
$IFWT(A) = [IFWT(A_0) - IFWT(A_1), IFWT(A_1)]$

85.3 异或运算

令 $d(x)$ 为 x 在二进制下拥有的 1 的数量
$FWT(A)[i] = \sum_j (-1)^{d(i\&j)} A[j]$
$FWT(A) = [FWT(A_0 + A_1), FWT(A_0 - A_1)]$
$IFWT(A) = [\frac{IFWT(A_1 - A_0)}{2}, \frac{IFWT(A_1 + A_0)}{2}]$

85.4 code

```
namespace FWT {
#define forforfor for (int l = 2; l <= len; l <<= 1)\
                  for (int i = 0, k = l>>1; i < len; i += l)\
                  for (int j = 0; j < k; ++j)

  const int SIZE = (1<<17)+3;
  int len;
  int f[SIZE], g[SIZE];
  template <class T> void init(T a[], const int &n, T b[], const int &m) {
    len = 1;
    while (len < max(n, m)) len <<= 1;
    for (int i = 0; i < n; ++i) f[i] = a[i];
    for (int i = n; i < len; ++i) f[i] = 0;
    for (int i = 0; i < m; ++i) g[i] = b[i];
    for (int i = m; i < len; ++i) g[i] = 0;
  }
  template <class T> void fwt_or(T a[], const int x = 1) {
    forforfor a[i+j+k] = (a[i+j+k]+1ll*a[i+j]*x)%MOD;
  }
  template <class T> void fwt_and(T a[], const int x = 1) {
    forforfor a[i+j] = (a[i+j]+1ll*a[i+j+k]*x)%MOD;
  }
  template <class T> void fwt_xor(T a[], const int x = 1) {
    forforfor {
      (a[i+j] += a[i+j+k]) %= MOD;
      a[i+j+k] = (a[i+j]-2*a[i+j+k]%MOD+MOD)%MOD;
      a[i+j] = 1ll*a[i+j]*x%MOD; a[i+j+k] = 1ll*a[i+j+k]*x%MOD;
    }
  }
  template <class T> void work_or(const T a[], const int &n, const T b[], const int &m) {
    init(a, n, b, m); fwt_or(f); fwt_or(g);
    for (int i = 0; i < len; ++i) f[i] = 1ll*f[i]*g[i]%MOD;
    fwt_or(f, MOD-1); // fwt_or(x, -1)
  }
  template <class T> void work_and(const T a[], const int &n, const T b[], const int &m) {
```

```
    init(a, n, b, m); fwt_and(f); fwt_and(g);
    for (int i = 0; i < len; ++i) f[i] = 1ll*f[i]*g[i]%MOD;
    fwt_and(f, MOD-1); // fwt_and(x, -1)
  }
  template <class T> void work_xor(const T a[], const int &n, const T b[], const int &m) {
    init(a, n, b, m); fwt_xor(f); fwt_xor(g);
    for (int i = 0; i < len; ++i) f[i] = 1ll*f[i]*g[i]%MOD;
    fwt_xor(f, mul_inverse(2)); // fwt_xor(x, 1/2)
  }
#undef forforfor
} // namespace FWT
```

86 快速子集变换 (子集卷积)|FST

$$C_k = \sum_{i\&j=0, i|j=k} A_i B_j$$

复杂度 $O(n \log^2 n) | O(n^2 2^n)$

```
namespace FST {
  const int W = 20;
  const int N = 1<<W;
  int len, bit;
  int f[W+1][N], g[W+1][N], h[W+1][N], res[N];
  template <class T> void fwt(T a[], const int x = 1) {
    for (int l = 2; l <= len; l <<= 1)
    for (int i = 0, k = l>>1; i < len; i += l)
    for (int j = 0; j < k; ++j)
      a[i+j+k] = (a[i+j+k]+1ll*a[i+j]*x)%MOD;
  }
  template <class T> void work(const T a[], const int &n, const T b[], const int &m) {
    len = 1; bit = 0;
    while (len < max(n, m)) len <<= 1, ++bit;
    for (int i = 0; i <= bit; ++i)
      for (int j = 0; j < len; ++j)
        f[i][j] = g[i][j] = h[i][j] = 0;
    for (int i = 0; i < n; ++i) f[__builtin_popcount(i)][i] = a[i];
    for (int i = 0; i < m; ++i) g[__builtin_popcount(i)][i] = b[i];
    for (int i = 0; i <= bit; ++i) {
      fwt(f[i]); fwt(g[i]);
      for (int j = 0; j <= i; ++j)
        for (int k = 0; k < len; ++k)
          h[i][k] = (h[i][k]+1ll*f[j][k]*g[i-j][k])%MOD;
      fwt(h[i], MOD-1); // fwt(h[i], -1)
    }
    for (int i = 0; i < len; ++i) res[i] = h[__builtin_popcount(i)][i];
  }
} // namespace FST
```

86.1 倍增子集卷积

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

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

按照每个状态的最高位进行分组，然后卷 n 次

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

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

87 第二类斯特林数

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

```
void stirling(const int &n) {
  inv[0] = inv[1] = 1;
  for(int i = 2; i <= n; ++i)
    inv[i] = MOD-MOD/i*inv[MOD%i]%MOD;
  for (int i = 1; i <= n; ++i)
    inv[i] = inv[i-1]*inv[i]%MOD;
  while (len <= (n<<1)) len <<= 1, ++bit;
  for (int i = 0; i < len; ++i)
    rev[i] = (rev[i>>1]>>1)|((i&1)<<(bit-1));
  for (int i = 0, one = 1; i <= n; ++i, one = MOD-one) {
    f[i] = one*inv[i]%MOD;
    g[i] = qpow(i, n)*inv[i]%MOD;
  }
  NTT(f, 1); NTT(g, 1);
  for (int i = 0; i < len; ++i) f[i] = f[i]*g[i]%MOD;
  NTT(f, -1);
  long long invv = qpow(len, MOD-2);
  for (int i = 0; i <= n; ++i)
    printf("%lld%c", f[i]*invv%MOD, " \n"[i==n]);
}
```

88 约瑟夫环

88.1 O(n)

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

89 最小公倍数 lcm

$$LCM(\frac{a}{b},\frac{c}{d}) = \frac{LCM(a,c)}{GCD(b,d)}$$
$$LCM(\frac{a_1}{b_1},\frac{a_2}{b_2},...) = \frac{LCM(a1,a2,...)}{GCD(b1,b2,...)}$$

90 扩展欧几里得 (同余方程)

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

91 乘法逆元

91.1 拓展欧几里得

```
template <typename T>
inline T mul_inverse(const T &a, const T &mo = MOD) {
  T x, y;
  exgcd(a, mo, x, y);
  return (x%mo+mo)%mo;
}
```

91.2 线性递推

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

92 中国剩余定理

92.1 中国剩余定理 CRT(m 互质)

```
inline long long CRT(int a[], int m[]) {
  long long res = 0, M = 1;
  for (int i = 1; i <= n; ++i)
    M *= m[i];
  for (int i = 1; i <= n; ++i)
    res = (res + a[i]*(M/m[i])*mul_inverse(M/m[i], m[i]))%M;
  return (res+M)%M;
}
```

92.2 扩展中国剩余定理 EXCRT(m 不互质)

```
inline long long EXCRT(long long a[], long long m[]) {
  // M*x + m[i]*y = a[i]-res (mod m[i])
  // res = res+x*M;
  long long M = m[1], res = a[1], x, y, c, d;
  for (int i = 2; i <= n; ++i) {
    d = exgcd(M, m[i], x, y);
    c = (a[i]-res%m[i]+m[i])%m[i];
    if (c%d != 0) return -1;
    x = (c/d)*x%(m[i]/d);
    res += x*M;
    M *= m[i]/d;
    res = (res%M+M)%M;
  }
  return res;
}
```

93 排列组合

93.1 奇偶性

C(n,k) 当 n&k == k 为 `奇数反之偶数

94 欧拉函数

```
template <typename T> inline T phi(T x) {
  T res = x;
  for (T i = 2; i*i <= x; ++i) {
    if (x%i) continue;
    res = res/i*(i-1);
    while (x%i == 0) x /= i;
  }
  if (x > 1) res = res/x*(x-1);
  return res;
}
```

94.1 筛法

```cpp
struct Euler {
  int phi[N], check[N];
  vector<int> prime;
  void init(int sz) {
    for (int i = 1; i <= sz; ++i) check[i] = 1;
    phi[1] = 1; check[1] = 0;
    for (int i = 2; i <= sz; ++i) {
      if (check[i]) {
        prime.emplace_back(i);
        phi[i] = i-1;
      }
      for (int j : prime) {
        if (i*j > sz) break;
        check[i*j] = 0;
        if (i%j) {
          phi[i*j] = (j-1)*phi[i];
        } else {
          phi[i*j] = j*phi[i];
          break;
        }
      }
    }
  }
} E;
```

94.2 欧拉定理

a 与 m 互质时，$a^{\phi(m)} \equiv 1 \mod m$

94.3 扩展欧拉定理

无需 a,m 互质 $b > \phi(m), a^b \equiv a^{(b \mod \phi(m))+\phi(m)} \mod m$

95 莫比乌斯函数

```cpp
template <typename T> inline T miu(T x) {
  int t = 0;
  for (T i = 2, k; i*i <= x; ++i) {
    if (x%i) continue;
    for (k = 0, ++t; x %i == 0; x /= i, ++k) {}
    if (k >= 2) return 0;
  }
  if (x > 1) ++t;
  return t&1 ? -1 : 1;
}
```

96 线性筛素数

```cpp
struct Euler {
  vector<int> prime;
  vector<bool> check;
  bool operator  { return check[i]; }
  void init(int n) {
    prime.clear();
    check = vector<bool>(n+1, true);
    check[1] = false;
    for (int i = 2; i <= n; ++i) {
      if (check[i]) prime.emplace_back(i);
      for (const int &j : prime) {
        if (i*j > n) break;
        check[i*j] = false;
        if (i%j == 0) break;
      }
    }
  }
} E;
```

97 判断素数（质数）

97.1 Miller-Rabin 素性测试

```cpp
inline bool MillerRabin(int x) {
  static const int test_time = 10;
  if (x < 3) return x == 2;
  int a = x-1, b = 0;
  while (!(a&1)) a >>= 1, ++b;
  for (int i = 1, j, v; i <= test_time; ++i) {
    v = (qpow(rnd()%(x-2)+2, a, x));
    if (v == 1 || v == x-1) continue;
    for (j = 0; j < b && v != x-1; ++j)
      v = static_cast<int>(1ll*v*v%x);
    if (j >= b) return false;
  }
  return true;
}
```

98 线性筛 GCD

```cpp
inline void gcd_init(const int &n) {
  for (int i = 1; i <= n; ++i)
  for (int j = 1; j <= n; ++j) if (!g[i][j])
  for (int k = 1; k <= n/i; ++k)
    g[k*i][k*j] = k;
}
```

99 BSGS

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

```cpp
inline long long BSGS(long long a, long long x, long long m) { // a^n = x
  static map<long long, int> mmp; mmp.clear();
  long long t = sqrt(m)+1, b = 1, c = 1, res;
  for(int i = 0; i < t; ++i, b = b*a%m) mmp[x*b%m] = i;
  for(int i = 0; i <= t; ++i, c = c*b%m) { // b = a^t
    if(mmp.count(c)) {
      res = t*i-mmp[c];
      if(res > 0) return res;
    }
  }
  return -1;
}
```

100 拓展 BSGS

$\gcd(a, m) \neq 1$

```cpp
namespace EXBSGS {

inline ll BSGS(ll a, ll b, ll mod, ll qaq){
  unordered_map<ll, int> H; H.clear();
  ll Q, p = ceil(sqrt(mod)), x, y;
  exgcd(qaq, mod, x, y), b = (b * x % mod + mod) % mod,
  Q = qpow(a, p, mod), exgcd(Q, mod, x, y), Q = (x % mod + mod) % mod;
  for (ll i = 1, j = 0 ; j <= p ; ++ j, i = i * a % mod)  if (!H.count(i)) H[i] = j ;
  for (ll i = b, j = 0 ; j <= p ; ++ j, i = i * Q % mod)  if (H[i]) return j * p + H[i];
  return -1 ;
}

inline ll exBSGS(ll N, ll P, ll M){
  ll qaq = 1, k = 0, qwq = 1;
  if (M == 1) return 0 ;
  while ((qwq = __gcd(N, P)) > 1) {
    if (M%qwq) return -1 ;
    ++k, M /= qwq, P /= qwq, qaq = qaq*(N/qwq)%P ;
    if (qaq == M) return k ;
  }
  return (qwq = BSGS(N, M, P, qaq)) == -1 ? -1 : qwq+k ;
}

} using EXBSGS::exBSGS;
```

101 错排

$D_1 = 0$
$D_2 = 1$
$D_n = (n-1)(D_{n-1} + D_{n-2})$

102 原根

复杂度 $O(\sqrt{m}+g\times\log^2 m)$

```cpp
inline int getG(const int &m) {
  static int q[100000+7];
  int _phi = phi(m), x = _phi, tot = 0;
  for (int i = 2; i*i <= _phi; ++i) {
    if (x%i) continue;
    q[++tot] = _phi/i;
    while (x%i == 0) x /= i;
  }
  if (x > 1) x = q[++tot] = _phi/x;
  for (int g = 2, flag; ; ++g) {
    flag = 1;
    if (qpow(g, _phi, m) != 1) continue;
    for (int i = 1; i <= tot; ++i) {
      if (qpow(g, q[i], m) == 1) {
        flag = 0;
        break;
      }
    }
    if (flag) return g;
  }
}
```

102.1 单位根反演

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

$$[a\equiv b(\mod n)]=[a-b\equiv 0(\mod n)]=\frac{1}{n}\sum_{i=0}^{n-1}w_n^{(a-b)k}=\frac{1}{n}\sum_{i=0}^{n-1}w_n^{ak}w_n^{-bk}$$

102.2 单位根卷积

$$\sum_{i=0}^{n}[i\%k=0]f(i)=\sum_{i=0}^{n}\frac{1}{k}\sum_{j=0}^{k-1}(w_k^i)^j f(i)$$

103 全排列和逆序对

103.1 根据逆序数推排列数

已知一个 n 元排列的逆序数为 m，这样的 n 元排列有多少种？

1. 对任意 n>=2 且 0<=m<=C(n,2) 时 f(n,m)>=1; 当 m>C(n,2) 时,f(n,m)=0
2. f(n,m)=f(n,C(n,2)-m)
3. f(n+1,m)=f(n,m)+f(n,m-1)+…+f(n,m-n)
4. f(n,0)=f(n,C(n,2))=1
5. f(n,1)=f(n,C(n,2)-1)=n-1(n>1)
6. f(n,2)=f(n,C(n,2)-2)=C(n,2)-1(n>2)

103.2 根据每个数的逆序数求出原排列

103.3 根据逆序数求最小排列

1. 对于 n 的全排列，在它完全倒序的时候（也就是 n,n-1,…,2,1 的时候）逆序数最多。
2. 对于一个形如 1,2,3,…,i-1,i,n,…i+1 的排列 q（如 n=5 时的 1,2,5,4,3），即在数 n 前保证首项为 1 且严格以公差为 1 递增而数 n 之后排列任意的数列
   - 当数 n 之后是递减的时候 q 的逆序数最多，为 t=C(n-i,2)。
   - 排列 q 是出现逆序数为 t 的最小排列。
3. 在上一条所设定的排列 q 的基础上，我们将数 n 后面的第 k 小数与数 n 的前一个数（即 i）交换，然后使数 n 后面保持逆序。这样得到的新排列所含的逆序数为 t=C(n-i,2)+k，且这个排列是逆序数为 t 的最小排列。

103.4 第 k 个字典序每个数的逆序对

```
// n个数排列的第k个(字典序)的逆序对
int f(int k) {
  int res = 0;
  for (int j = n; j; --j) {
    res += k/fac[j];
    k %= fac[j];
  }
  return res;
}
```

104 二次剩余

```
namespace Cipolla {

int mod, _x0, _x1;
long long I_mul_I; // 虚数单位的平方

struct complex {
  long long real, imag;
  complex(long long real = 0, long long imag = 0): real(real), imag(
      imag) { }
};
inline bool operator == (complex x, complex y) {
  return x.real == y.real and x.imag == y.imag;
}
inline complex operator * (complex x, complex y) {
  return complex((x.real * y.real + I_mul_I * x.imag % mod * y.imag)
      % mod,
       (x.imag * y.real + x.real * y.imag) % mod);
}

complex power(complex x, int k) {
  complex res = 1;
  while(k) {
    if(k & 1) res = res * x;
    x = x * x;
    k >>= 1;
  }
```

```
  return res;
}

bool check_if_residue(int x) {
  return power(x, (mod - 1) >> 1) == 1;
}

int solve(int n, int p, int &x0 = _x0, int &x1 = _x1) {
  mod = p;
  if (power(n, p>>1) == p-1) return x0 = -1; // 无解
  if (power(n, p>>1) == 0) return x0 = x1 = 0;

  long long a = rand() % mod;
  while(!a or check_if_residue((a * a + mod - n) % mod))
    a = rand() % mod;
  I_mul_I = (a * a + mod - n) % mod;

  x0 = int(power(complex(a, 1), (mod + 1) >> 1).real);
  x1 = mod - x0;
  return x0;
}
} // namespace Cipolla
```

Part VII
动态规划 DP

105 线性 DP

105.1 最长公共子序列 LCS

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

106 状压 DP

106.1 枚举子集

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

106.2 枚举 n 个元素，大小为 k 的二进制子集

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

107 背包问题

107.1 多重背包

二进制拆分

```
for(int i = 1, cnt, vi, wi, m; i <= n; ++i) {
  scanf("%d%d%d", &vi, &wi, &m);
  cnt = 1;
  while(m-cnt > 0) {
    m -= cnt;
    v.push_back(vi*cnt);
    w.push_back(wi*cnt);
    cnt <<= 1;
  }
  v.push_back(vi*m);
  w.push_back(wi*m);
}
for(int i = 0; i < w.size(); ++i)
  for(int j = W; j >= w[i]; --j)
    b[j] = max(b[j], b[j-w[i]]+v[i]);
```

单调队列

```
for(int i = 1; i <= n; ++i) {
  scanf("%d%d%d", &v, &w, &m);
  for(int u = 0; u < w; ++u) {
    int maxp = (W-u)/w;
    head = 1; tail = 0;
    for(int k = maxp-1; k >= max(0, maxp-m); --k) {
      while(head <= tail && calc(u, q[tail]) <= calc(u, k)) tail--;
      q[++tail] = k;
    }
    for(int p = maxp; p >= 0; --p) {
      while(head <= tail && q[head] >= p) head++;
      if(head <= tail) f[u+p*w] = max(f[u+p*w], p*v+calc(u, q[head]))
        ;
      if(p-m-1 < 0) continue;
      while(head <= tail && calc(u, q[tail]) <= calc(u, p-m-1)) tail
        --;
      q[++tail] = p-m-1;
    }
  }
}
int ans = 0;
for(int i = 1; i <= W; ++i)
  ans = max(ans, f[i]);
```

108 SOS DP

109 WQS 二分 |DP 凸优化

题目给了一个选物品的限制条件，要求刚好选 m 个，让你最大化（最小化）权值，其特点是函数的斜率单调

例：给你一个 N 个点 M 条边无向带权连通图，每条边是黑色或白色。让你求一棵最小权的恰好有 K 条白色边的生成树。

记 $g(i)$ 是选了 i 条白边的最小生成树值，发现 $g(i)$ 斜率单调不增 $g(i)-g(i-1) \le g(i+1)-g(i)$

则二分斜率 k，求切点（截距最大）

设 $f(x)$ 为我在没有固定选多少个点（但是我已经选了 x 个点）时的答案（也就是截距），$f(x)=g(x)-k*x$

只要把每个数的 $h(x)-=k$ 然后正常求一下在选任意个数的情况下最大 $f(x)$ 是多少 $O(n\log n)$

110 斜率优化

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

110.1 「HNOI2008」玩具装箱 TOY

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

令 $a[i]=sum[i]+i, b[i]=sum[i]+i+L+1$

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

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

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

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

而对于每个 i 来说，$a[i]$ 都是确定的，类似线性规划

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

111 四边形不等式

111.1 2D1D

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

当 $w(l,r)$ 满足特定性质

- 区间包含单调性：如果对于任意 $l\le l'\le r'\le r$ ，均有 $w(l',r')\le w(l,r)$ 成立，则称函数 w 对于区间包含关系具有单调性。
- 四边形不等式：如果对于任意 $l_1\le l_2\le r_1\le r_2$ ，均有 $w(l_1,r_1)+w(l_2,r_2)\le w(l_1,r_2)+w(l_2,r_1)$ 成立，则称函数 w 满足四边形不等式（简记为“交叉小于包含”）。若等号永远成立，则称函数 w 满足四边形恒等式。

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

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

111.2 1D1D

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

> 定理 2：若函数 $w(l,r)$ 满足四边形不等式，记 $h_{l,r}=f_l+w(l,r)$ 表示从 l 转移过来的状态 r ，$k_r=\min\{l|f_r=h_{l,r}\}$ 表示最优决策点，则有 $\forall r_1\le r_2: k_{r1}\le k_{r2}$

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

111.3 满足四边形不等式的函数类

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

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

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

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

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

112 插头 DP| 轮廓线 DP

112.1 一个闭合回路

```
const int P = 299987;
const int M = 1<<21;
const int N = 15;

int n, m;
int a[N][N];
long long dp[2][M];
int head[2][P], nex[2][M], tot[2], ver[2][M];
// long long dp[2][P];
// int head[2][P], nex[2][P], tot[2], ver[2][P];

inline void clear(const int &u) {
  for (int i = 1; i <= tot[u]; ++i) {
    dp[u][i] = 0; //
    nex[u][i] = 0; //
    head[u][ver[u][i]%P] = 0;
  }
  tot[u] = 0;
}

template <typename T, typename U>
inline void insert(const int &u, const T &x, const U &v) {
  int p = x%P;
  for (int i = head[u][p]; i; i = nex[u][i]) {
```

```
    if (ver[u][i] == x) return dp[u][i] += v, void();
  }
  ++tot[u]; assert(tot[u] < M);
  ver[u][tot[u]] = x;
  nex[u][tot[u]] = head[u][p];
  head[u][p] = tot[u];
  dp[u][tot[u]] = v;
}

template <typename T>
inline int get_val(const int &u, const T &x) {
  int p = x%P;
  for (int i = head[u][p]; i; i = nex[u][i]) {
    if (ver[u][i] == x) return dp[u][i];
  }
  return 0;
}

inline long long solve() {
  int u = 0, base = (1<<m*2+2)-1;
  long long res = 0;
  clear(u);
  insert(u, 0, 1);
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      clear(u ^= 1);
      for (int k = 1; k <= tot[u^1]; ++k) {
        int state = ver[u^1][k];
        long long val = dp[u^1][k];
        if (j == 1) state = (state<<2)&base;
        // b1 right b2 down
        // 0 no 1 left 2 right
        int b1 = (state>>j*2-2)%4, b2 = (state>>j*2)%4;
        if (!a[i][j]) {
          if (!b1 && !b2) insert(u, state, val);
        } else if (!b1 && !b2) {
          if (a[i+1][j] && a[i][j+1]) insert(u, state+(1<<j*2-2)+(2<<
              j*2), val);
        } else if (!b1 && b2) {
          if (a[i][j+1]) insert(u, state, val);
          if (a[i+1][j]) insert(u, state+(b2<<j*2-2)-(b2<<j*2), val);
        } else if (b1 && !b2) {
          if (a[i+1][j]) insert(u, state, val);
          if (a[i][j+1]) insert(u, state-(b1<<j*2-2)+(b1<<j*2), val);
        } else if (b1 == 1 && b2 == 1) { // find 2 turn to 1
          for (int k = j+1, t = 1; k <= m; ++k) {
            if ((state>>k*2)%4 == 1) ++t;
            if ((state>>k*2)%4 == 2) --t;
            if (!t) { insert(u, state-(1<<j*2-2)-(1<<j*2)-(1<<k*2),
                val); break; }
          }
        } else if (b1 == 2 && b2 == 2) { // find 1 turn to 2
          for (int k = j-2, t = 1; k >= 0; --k) {
            if ((state>>k*2)%4 == 1) --t;
            if ((state>>k*2)%4 == 2) ++t;
            if (!t) { insert(u, state-(2<<j*2-2)-(2<<j*2)+(1<<k*2),
                val); break; }
```

```
          }
        } else if (b1 == 2 && b2 == 1) {
          insert(u, state-(2<<j*2-2)-(1<<j*2), val);
        } else if (i == ex && j == ey) { // b1 == 1, b2 == 2
          res += val;
        }
      }
    }
  }
  return res;
}
```

112.2 多个闭合回路

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

112.3 联通块

```
int n, u, res = -INF;
int a[N][N];
unordered_map<int, int> dp[2];

inline void decode(const int &state, int *const s) {
  for (int i = 1; i <= n; ++i) s[i] = (state>>i*3-3)%8;
}

inline void insert(const int *const s, const int &val) {
  static int vis[N];
  int state = 0, cnt = 0;
  memset(vis, 0, sizeof vis);
  for (int i = 1; i <= n; ++i) {
    if (s[i] && !vis[s[i]]) vis[s[i]] = ++cnt;
    state |= (vis[s[i]]<<i*3-3);
  }
  if (dp[u].count(state)) dp[u][state] = max(dp[u][state], val);
  else dp[u].insert({state, val});
  if (cnt == 1) res = max(res, val);
}

inline void solve() {
  static int s[N];
  dp[u = 0].clear();
  dp[u][0] = 0;
  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= n; ++j) {
      dp[u ^= 1].clear();
      for (const auto &p : dp[u^1]) {
        decode(p.first, s);
        int b1 = s[j-1], b2 = s[j];
        // not choose
```

```
33          s[j] = 0;
34          int cnt = 0;
35          for (int k = 1; k <= n; ++k) cnt += s[k] == b2;
36          if (!b2 || cnt) insert(s, p.second);
37          s[j] = b2;
38          // choose
39          if (!b1 && !b2) {
40            s[j] = 7;
41          } else {
42            if (b1 > b2) swap(b1, b2); // in case b2 == 0
43            s[j] = b2;
44            if (b1) for (int k = 1; k <= n; ++k) if (s[k] == b1) s[k] =
                  b2;
45          }
46          insert(s, p.second+a[i][j]);
47        }
48      }
49    }
50    cout << res << endl;
51  }
```

112.4 L 型

L 型地板：拐弯且仅拐弯一次。

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

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

113 DP 套 DP

有 $dp_1, f[i]$ 为 $dp_1[n] = i$ 的方案数，求 f

设 $dp_2[dp_1]$ 为 dp_1 状态下的方案数

114 动态 DP

将 dp 转换为线段树可以求解的区间问题，动态维护

114.1 动态线性 DP

114.2 动态树形 DP

树链剖分，轻链暴力

Part VIII
分数

```
1  template <class T>
2  struct Fraction {
3    T a, b;
```

```
  void normalize() {
    if (a == 0) return (void)(b = 1);
    T g = __gcd(a, b);
    a /= g; b /= g;
    if (b < 0) a = -a, b = -b;
  }
  Fraction(const T &_a = 0, const T &_b = 1) : a(_a), b(_b) {
     normalize(); }
  friend bool operator < (const Fraction &f1, const Fraction &f2) {
     return f1.a*f2.b < f2.a*f1.b; }
  friend bool operator > (const Fraction &f1, const Fraction &f2) {
     return f1.a*f2.b > f2.a*f1.b; }
  friend bool operator == (const Fraction &f1, const Fraction &f2) {
     return f1.a*f2.b == f2.a*f1.b; }
  friend bool operator != (const Fraction &f1, const Fraction &f2) {
     return f1.a*f2.b != f2.a*f1.b; }
  friend bool operator <= (const Fraction &f1, const Fraction &f2) {
     return f1 < f2 || f1 == f2; }
  friend bool operator >= (const Fraction &f1, const Fraction &f2) {
     return f1 > f2 || f1 == f2; }
  friend Fraction operator + (const Fraction &f1, const Fraction &f2)
      { return Fraction(f1.a*f2.b+f2.a*f1.b, f1.b*f2.b); }
  friend Fraction operator - (const Fraction &f1, const Fraction &f2)
      { return Fraction(f1.a*f2.b-f2.a*f1.b, f1.b*f2.b); }
  friend Fraction operator * (const Fraction &f1, const Fraction &f2)
      { return Fraction(f1.a*f2.a, f1.b*f2.b); }
  friend Fraction operator / (const Fraction &f1, const Fraction &f2)
      { return Fraction(f1.a*f2.b, f1.b*f2.a); }
  Fraction& operator += (const Fraction &f) { return *this = *this+f;
      }
  Fraction& operator -= (const Fraction &f) { return *this = *this-f;
      }
  Fraction& operator *= (const Fraction &f) { return *this = *this*f;
      }
  Fraction& operator /= (const Fraction &f) { return *this = *this/f;
      }
  friend istream& operator >> (istream &is, Fraction &f) {
    is >> f.a >> f.b;
    f.normalize();
    return is;
  }
  friend ostream& operator << (ostream &os, const Fraction &f) {
    if (f.b == 1) return os << f.a;
    return os << f.a << "/" << f.b;
  }
};
using fraction = Fraction<long long>;
```