目录

Part I 杂项

1 快读快写

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

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

2 玄学优化

吸氧，吸臭氧

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

3 正则表达式

```
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
```

4 随机数

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

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();}
```

5 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;
```

6 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;
}
```

7 i/k == j 的 k 的个数

r-l+1

```
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;
  }
}
```

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

Part II
计算几何

9 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; }
  template <typename TT>
```

```
  Point operator /= (const TT &k) { return *this = *this/k; }
  friend bool operator < (const Point &p1, const Point &p2) {
    return make_pair(p1.x, p1.y) < make_pair(p2.x, p2.y);
  }
  friend bool operator > (const Point &p1, const Point &p2) {
    return make_pair(p1.x, p1.y) > make_pair(p2.x, p2.y);
  }
  friend bool operator == (const Point &p1, const Point &p2) {
    return p1.x == p2.x && p1.y == p2.y;
  }
  friend bool operator != (const Point &p1, const Point &p2) {
    return p1.x != p2.x || p1.y != p2.y;
  }
  friend istream& operator >> (istream &is, Point &p) {
    return is >> p.x >> p.y;
  }
  friend ostream& operator << (ostream &os, Point &p) {
    return os << p.x << " " << p.y << " " << p.id << endl;
  }
  double length() { return sqrt(1.0*x*x+1.0*y*y); }
  friend double dis(const Point &p1, const Point &p2) { return (p2-p1).length(); }
  double dis(const Point &p) { return (p-*this).length(); }
  friend T dot(const Point &p1, const Point &p2) { return p1.x*p2.x+p1.y*p2.y; }
  T dot(const Point &p) { return x*p.x+y*p.y; }
  friend Point rotate_90_c(const Point &p) { return Point(p.y, -p.x, p.id); }
  Point rotate_90_c() { return Point(y, -x, id); }
  friend double atan(const Point &p) { return atan2(p.y, p.x); }
};

template <typename T = double>
struct Vec { // 三维向量
  T x, y, z;
  Vec(const T &_x = 0, const T &_y = 0, const T &_z = 0) : x(_x), y(_y), z(_z) {}
  double len() { return sqrt(1.0*x*x+1.0*y*y+1.0*z*z); }
  friend Vec operator +(const Vec &v1, const Vec &v2) { return Vec(v1.x+v2.x, v1.y+v2.y, v1.z+v2.z); }
  friend Vec operator -(const Vec &v1, const Vec &v2) { return Vec(v1.x-v2.x, v1.y-v2.y, v1.z-v2.z); }
  friend Vec operator *(const T &k, const Vec &v) { return Vec(k*v.x, k*v.y, k*v.z); }
  friend Vec operator *(const Vec &v, const T &k) { return k*v; }
  friend Vec operator *(const Vec &v1, const Vec &v2) {
    return Vec(
        v1.y*v2.z-v1.z*v2.y,
        v1.z*v2.x-v1.x*v2.z,
        v1.x*v2.y-v1.y*v2.x
    );
  }
  friend T dot(const Vec &v1, const Vec &v2) { return v1.x*v2.x+v1.y*v2.y+v1.z*v2.z; }
  T dot(const Vec &v) { return dot(*this, v); }
  Vec& operator +=(const Vec &v) { return *this = *this+v; }
  Vec& operator -=(const Vec &v) { return *this = *this-v; }
```

```cpp
  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)) < err;
  }
  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) < err) 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;
}
```

10 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;
  }
}
```

11 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);
}
```

12 最小圆覆盖 | 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

13 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;
    }
  }
};
```

14 平衡树

14.1 Splay

```cpp
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]; }
  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;
  struct node {
    T v = 0;
    int ch[2] = { 0, 0 };
    int fa = 0, sum = 0, cnt = 0, tag = 0;
  } e[N];
  int sz, &root = e[0].ch[1];
  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]; }
  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 = 0) {
    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) {
```

```
    ++sz; e[sz].v = v; e[sz].fa = fa; e[sz].sum = e[sz].cnt = 1;
    return sz;
  }
  int find(int x) {
    if (x > e[root].sum) return -INF;
    int now = root;
    while (true) {
      push_down(now);
      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];
    }
    return now;
  }
  int build(int l, int r, int fa) {
    if (l > r) return 0;
    int mid = (l+r)>>1,
      now = add_point(mid, fa);
    e[now].ch[0] = build(l, mid-1, now);
    e[now].ch[1] = build(mid+1, r, now);
    update(now);
    return now;
  }
  void push_down(int x) {
    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;
```

15 李超线段树

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

16 吉老师线段树 | 吉司机线段树

区间最值操作 & 区间历史最值
栗子：给出一个数列，每次操作让某个区间中对给定值取 min 询问某个区间的和

17 树套树

在第一维线段树的每个结点建立第二维线段树

18 树状数组

18.1 一维

单点修改区间查询
区间修改单点查询

```
template <typename T>
struct BinaryIndexedTree {
  int n;
  T tr[N];
  BinaryIndexedTree() { memset(tr, 0, sizeof tr); }
  void init(const int &_n) { n = _n; clear(); }
  void clear() { memset(tr+1, 0, sizeof(T)*n); }
  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); }
};
```

O(n) 初始化

```
template <typename TT>
void init(const int &_n, const TT a[]) {
  n = _n; clear();
  for (int i = 1; i <= n; ++i) {
    tr[i] += a[i];
    if (i+(i&-i) <= n) tr[i+(i&-i)] += tr[i];
  }
}
```

18.2 二维

18.2.1 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;
  }
};
```

19 可持久化线段树（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);
  }
};
```

20 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;
  }
  bool query(const int &cur, const int &x, const int &y) {
    return fa[getf(rt[cur], x)] == fa[getf(rt[cur], y)];
  }
  // return the id of fa[], dep[]
  int query(const int &cur, const int &x, const int &l, const int &r)
      {
    if (l == r) return cur;
    int mid = (l+r)>>1;
    if (x <= mid) return query(ls[cur], x, l, mid);
    else return query(rs[cur], x, mid+1, r);
  }
  // return the id of fa[], dep[]
  int getf(const int &cur, int x) {
    int fi;
    while (fa[(fi = query(cur, x, 1, n))] != x) x = fa[fi];
    return fi;
  }
  void merge(const int &cur, const int &pre, const int &x, const int
     &y) {
    rt[cur] = rt[pre];
```

```
    int fx = getf(rt[cur], x), fy = getf(rt[cur], y);
    if (fa[fx] == fa[fy]) return;
    if (dep[fx] > dep[fy]) swap(fx, fy);
    rt[cur] = update(rt[pre], fa[fx], fa[fy], 1, n);
    if (dep[fx] == dep[fy]) add(rt[cur], fa[fy], 1, n);
  }
  // update fa, merge x to y
  int update(const int &pre, const int &x, const int &y, const int &l
     , const int &r) {
    int cur = ++tot; assert(tot < NN);
    if (l == r) return fa[cur] = y, dep[cur] = dep[pre], cur;
    ls[cur] = ls[pre]; rs[cur] = rs[pre];
    int mid = (l+r)>>1;
    if (x <= mid) ls[cur] = update(ls[pre], x, y, l, mid);
    else rs[cur] = update(rs[pre], x, y, mid+1, r);
    return cur;
  }
  // add dep
  void add(const int &cur, const int &x, const int &l, const int &r)
     {
    if (l == r) return ++dep[cur], void();
    int mid = (l+r)>>1;
    if (x <= mid) add(ls[cur], x, l, mid);
    else add(rs[cur], x, mid+1, r);
  }
};
```

21 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);
  }
};
```

22 分块

```
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;
```

23 莫队

$O(1)$ 一般取 $block=\frac{n}{\sqrt{m}}, O(n\sqrt{m})$
移动前两步先扩大区间 $l--, r++$ 后两步缩小区间 $l++, r--$

23.1 奇偶性排序

```
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;
}
```

23.2 带修改莫队

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

```
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;
}
```

23.3 值域分块

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

```
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]; }
};
```

23.4 二次离线莫队

大概是一种需要维护信息具有可减性的莫队。只要具可减性，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 $O(n\sqrt{n}+n\sqrt{n})$

```
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() {
  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);
        else res[-qq.id] -= suml.query(a[j]+1);
      }
    }
  }
  sumr.init(n);
  for (int i = n; i; --i) {
    sumr.add(a[i], 1);
    for (auto &qq : qr[i]) {
      for (int j = qq.l; j <= qq.r; ++j) {
        if (qq.id > 0) res[qq.id] += sumr.query(a[j]-1);
```

```
      else res[-qq.id] -= sumr.query(a[j]-1);
    }
  }
}
for (int i = 1; i <= m; ++i) {
  res[i] += res[i-1];
  ans[q[i].id] = res[i];
}
for (int i = 1; i <= m; ++i) write(ans[i]), putchar('\n');
}
```

24 ST 表

24.1 一维

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

24.2 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]
```

```cpp
  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]));
  }
};
```

24.3 反向 ST

```cpp
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]);
  }
};
```

25 并查集

```
struct DSU {
  int fa[N];
  void init(int sz) { for (int i = 0; i <= sz; ++i) fa[i] = i; }
  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
    int fx = get(x), fy = get(y);
    if (fx == fy) return false;
    fa[fx] = fy; return true;
  }
} dsu;
```

加上数量

```
struct DSU {
  int fa[N], num[N];
  void init(int sz) { for (int i = 0; i <= sz; ++i) fa[i] = i, num[i]
     = 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) {
    int fx = get(x), fy = get(y);
    if (fx == fy) return false;
    if (num[fx] >= num[fy]) num[fx] += num[fy], fa[fy] = fx;
    else num[fy] += num[fx], fa[fx] = fy;
    return true;
  }
} dsu;
```

Part IV
字符串

26 回文字符串 |manacher 算法

从 0 开始，第 i 位对应 p[i*2+2]

```
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;
}
```

26.1 判断 s[l, r] 是否为回文

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

27 KMP

```
inline void get_next(const string &s, int nex[]) { get_next(s.c_str()
   , nex); }
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 string &s1, const string &s2, int nex[]) { kmp(
   s1.c_str(), s2.c_str(), nex); }
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];
    }
  }
}
```

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

```
  }
}

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

28 扩展 KMP|Z 函数

```
inline void GetNext(char *s, int *_nex) {
  int len = strlen(s);
  int a = 0, p = 0;
  _nex[0] = len;
  for (int i = 1; i < len; ++i) {
    if (i >= p || i+_nex[i-a] >= p) {
      if (i > p) p = i;
      while (p < len && s[p] == s[p-i]) ++p;
      a = i;
      _nex[i] = p-i;
    } else {
      _nex[i] = _nex[i-a];
    }
  }
}

inline void GetExtend(char *s, char *ss, int *_ext, int *_nex) {
  int lens = strlen(s), lenss = strlen(ss);
  int a = 0, p = 0;
  for (int i = 0; i < lens; ++i) {
    if (i >= p || i+_nex[i-a] >= p) {
      if (i > p) p = i;
      while (p < lens && p-i < lenss && s[p] == ss[p-i]) ++p;
      a = i;
      _ext[i] = p-i;
    } else {
      _ext[i] = _nex[i-a];
    }
  }
}
```

29 后缀数组 |SA

29.1 $O(nlog^2n)$

```cpp
int sa[N], rk[N<<1], height[N];
template <typename T> // s start from 1
inline void SA(const T *s, const int &n) {
  static int oldrk[N<<1];
  memset(rk+n+1, 0, sizeof(int)*n);
  for (int i = 1; i <= n; ++i) rk[i] = s[i];
  for (int w = 1; w <= n; w <<= 1) {
    iota(sa+1, sa+n+1, 1);
    sort(sa+1, sa+n+1, & {
      return rk[x] == rk[y] ? rk[x+w] < rk[y+w] : rk[x] < rk[y];
    });
    memcpy(oldrk+1, rk+1, sizeof(int)*2*n);
    for (int p = 0, i = 1; i <= n; ++i) {
      if (oldrk[sa[i]] == oldrk[sa[i-1]] &&
        oldrk[sa[i]+w] == oldrk[sa[i-1]+w]) {
        rk[sa[i]] = p;
      } else {
        rk[sa[i]] = ++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;
  }
}
```

29.2 $O(n)$

```cpp
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
```

30 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);
    memset(num, 0, sizeof num);
    memset(exist, 0, sizeof exist);
    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()); }
};
```

31 AC 自动机

如需构造可重建 AC 自动机，每次构造建一个 nex 数组的拷贝

```
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;
```

Part V
图论 | 树论

32 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]);
}
```

33 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;
```

34 稳定婚姻匹配

```cpp
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,
         &;
      //sort(rkg[i]+1, rkg[i]+n+1,
      //    &;
    }
  }
  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);
    }
  }
};
```

35 最小生成树

Prim

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

Kruskal (略)

36 二分图

36.1 二分图匹配

匈牙利算法

```
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;
}
```

36.2 二分图最小顶点覆盖

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

定理：最小顶点覆盖等于二分图的最大匹配。

36.3 最大独立集

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

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

37 最近公共祖先 |LCA

37.1 倍增

```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];
  }
};
```

37.2 带权 LCA

```cpp
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]));
  }
};
```

38 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];
    }
  }
};
```

39 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];
  SegmentTree<T> ST;
  template <typename Edge, typename TT>
  void build(const Edge e[], const TT a[], const int &n, const int &
     rt = 1) {
    fa[rt] = dfn = 0;
    dfs1(e, rt);
```

```
    dfs2(e, rt);
    for (int i = 1; i <= n; ++i)
      init_val[id[i]] = a[i];
    ST.build(init_val, n);
  }
  template <typename Edge>
  void dfs1(const Edge e[], 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(e, v);
      num[u] += num[v];
      if (num[v] > num[son[u]]) son[u] = v;
    }
  }
  template <typename Edge>
  void dfs2(const Edge e[], const int &u = 1) {
    tp[u] = son[fa[u]] == u ? tp[fa[u]] : u;
    id[u] = ++dfn;
    if (son[u]) dfs2(e, son[u]);
    for (const int &v : e[u]) if (v != son[u] && v != fa[u])
      dfs2(e, 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]);
  }
};
```

40 网络流

40.1 最大流

40.1.1 EK

$O(nm^2)$

```
template <typename T>
struct EK {
  struct Edge {
    int v, nex;
    T w;
  } e[M<<1];
  int tot = 0, n;
  int fir[N], vis[N], pre[N];
  T incf[N];
  T work(const int &s, const int &t) {
    T res = 0;
    while (bfs(s, t)) {
      int u = t, id;
      while (u != s) {
        id = pre[u];
        e[id].w -= incf[t];
        e[id^1].w += incf[t];
        u = e[id^1].v;
      }
      res += incf[t];
    }
    return res;
  }
  void init(const int &sz) {
    n = sz;
    tot = 0;
    memset(fir, -1, sizeof(int)*(n+3));
  }
  void add_edge(const int &u, const int &v, const T &w) {
    e[tot] = {v, fir[u], w}; fir[u] = tot++;
    e[tot] = {u, fir[v], 0}; fir[v] = tot++;
  }
  bool bfs(const int &s, const int &t) {
    queue<int> q;
    memset(vis, 0, sizeof(int)*(n+3));
    q.push(s);
    vis[s] = 1;
    incf[s] = INF;
    while (q.size()) {
      int u = q.front();
      q.pop();
      for (int i = fir[u], v; i != -1; i = e[i].nex) {
        v = e[i].v;
        if (vis[v] || !e[i].w) continue;
        incf[v] = min(incf[u], e[i].w);
        pre[v] = i;
        if (v == t) return true;
        q.push(v);
        vis[v] = 1;
      }
```

```
    }
    return false;
  }
};
```

40.1.2 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;
    EDGE(const int &_v, const int &_nex, const T &_w) : v(_v), nex(
       _nex), w(_w) {}
  };
  vector<EDGE> e;
  int n, s, t;
  int fir[N], dep[N], cur[N];
  Dinic() { e.reserve(N<<2); }
  T work(const int &_s, const int &_t) {
    s = _s; t = _t;
    T maxflow = 0, flow;
    while (bfs())
      while ((flow = dfs(s, INF)))
        maxflow += flow;
    return maxflow;
  }
  void init(const int &_n) {
    n = _n;
    e.clear();
    memset(fir, -1, sizeof(int)*(n+3));
  }
  void add_edge(const int &u, const int &v, const T &w) {
    e.emplace_back(v, fir[u], w); fir[u] = e.size()-1;
    e.emplace_back(u, fir[v], 0); fir[v] = e.size()-1;
  }
  bool bfs() {
    queue<int> q;
    memset(dep, 0, sizeof(int)*(n+3));
    q.push(s);
    dep[s] = 1;
    for (int i = 0; i <= n; ++i) cur[i] = fir[i];
    while (q.size()) {
      int u = q.front();
      q.pop();
      for (int i = fir[u], v; i != -1; i = e[i].nex) {
        v = e[i].v;
        if (dep[v] || !e[i].w) continue;
        dep[v] = dep[u]+1;
        if (v == t) return true;
        q.push(v);
      }
    }
    return false;
  }
```

```
  T dfs(const int &u, const T &flow) {
    if (!flow || u == t) return flow;
    T rest = flow, now;
    for (int &i = cur[u], v; i != -1; i = e[i].nex) {
      v = e[i].v;
      if (dep[v] != dep[u]+1 || !e[i].w) continue;
      now = dfs(v, min(rest, e[i].w));
      if (!now) {
        dep[v] = 0;
      } else {
        e[i].w -= now;
        e[i^1].w += now;
        rest -= now;
        if (rest == flow) break;
      }
    }
    return flow-rest;
  }
};
```

40.2 最小割

最小割等价最大流

40.3 费用流

40.3.1 MCMF

```
template <typename T>
struct MCMF {
  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], pre[N];
  T incf[N], dis[N];
  void init(const int &_n) {
    n = _n;
    e.clear();
    e.reserve(N<<4);
    memset(fir, -1, sizeof(int)*(n+3));
  }
  void add_edge(const int &u, const int &v, const T &w, const T &c) {
    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;
    T maxflow = 0, mincost = 0;
    while (spfa()) {
      for (int u = t, id; u != s; u = e[id^1].v) {
```

```
        id = pre[u];
        e[id].w -= incf[t];
        e[id^1].w += incf[t];
        mincost += incf[t]*e[id].c;
      }
      maxflow += incf[t];
    }
    return {maxflow, mincost};
  }
  bool spfa() {
    queue<int> q;
    memset(dis, 0x3f, sizeof(T)*(n+3));
    memset(vis, 0, sizeof(int)*(n+3));
    q.push(s);
    dis[s] = 0;
    incf[s] = INF;
    incf[t] = 0;
    while (q.size()) {
      int u = q.front();
      q.pop();
      vis[u] = 0;
      for (int i = fir[u], v; i != -1; i = e[i].nex) {
        v = e[i].v;
        if (!e[i].w || dis[v] <= dis[u]+e[i].c) continue;
        dis[v] = dis[u]+e[i].c;
        incf[v] = min(incf[u], e[i].w);
        pre[v] = i;
        if (vis[v]) continue;
        q.push(v);
        vis[v] = 1;
      }
    }
    return incf[t];
  }
};
```

40.3.2 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;
```

```
  }
};
```

41 最短路

41.1 Floyd

41.2 Dijkstra

41.3 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;
    }
  }
}
```

42 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;
```

```
}
```

43 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;
}
```

44 SCC 强连通分量 |Tarjan

```
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);
  }
}
```

45 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]);
    }
  }
}
```

46 2-SAT

46.1 SCC Tarjan

$O(n+m)$

```
struct TWO_SAT { // node stkrt 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);
    }
  }
};
```

46.2 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) {
```

```cpp
      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;
  }
};
```

47 虚树

```cpp
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;
}
```

48 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);
  }
};
```

49 矩阵树定理 |Kirchhoff

解决一张图的生成树个数计数问题 (详情见 oi-wiki)

Part VI 数论

50 快排

```
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);
}
```

51 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));
}
```

52 线性基

```
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;
  }
};
```

53 矩阵

```
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)*(n+1));
    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; }
  template <typename TT>
  Martix pow(const TT &p) const {
    Martix res(n, m), a = *this;
    res.init(1);
    for (TT i = p; i; i >>= 1, a *= a) if (i&1) res *= a;
    return res;
  }
  Martix inv() const {
    Martix res = *this;
    vector<int> is(n+1), js(n+1);
    for (int k = 1; k <= n; ++k) {
      for (int i = k; i <= n; ++i)
        for (int j = k; j <= n; ++j) if (res.a[i][j]) {
          is[k] = i; js[k] = j; break;
        }
      for (int i = 1; i <= n; ++i) swap(res.a[k][i], res.a[is[k]][i]);
      for (int i = 1; i <= n; ++i) swap(res.a[i][k], res.a[i][js[k]]);
      if (!res.a[k][k]) return Martix(0);
      res.a[k][k] = mul_inverse(res.a[k][k]); // get inv of number
      for (int j = 1; j <= n; ++j) if (j != k)
        res.a[k][j] = res.a[k][j]*res.a[k][k]%MOD;
      for (int i = 1; i <= n; ++i) if (i != k)
        for (int j = 1; j <= n; ++j) if (j != k)
          res.a[i][j] = (res.a[i][j]+MOD-res.a[i][k]*res.a[k][j]%MOD)%MOD;
      for (int i = 1; i <= n; ++i) if (i != k)
        res.a[i][k] = (MOD-res.a[i][k]*res.a[k][k]%MOD)%MOD;
    }
    for (int k = n; k; --k) {
      for (int i = 1; i <= n; ++i) swap(res.a[js[k]][i], res.a[k][i]);
      for (int i = 1; i <= n; ++i) swap(res.a[i][is[k]], res.a[i][k]);
    }
    return res;
  }
  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, Martix<T> &mx) {
  for (int i = 1; i <= mx.n; ++i)
    for (int j = 1; j <= mx.m; ++j)
      os << mx[i][j] << " \n"[j==mx.m];
  return os;
}
};
```

54 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;
  }
};
```

54.1 异或方程组

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

```
// -1 : 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]) return -a[i][n+1];
  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;
  }
}
```

55 拉格朗日插值

```
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;
}
```

```
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;
}
```

56 快速乘

```
inline long long qmul(long long x, long long y, long long mo) {
  long long res = 0;
  while (y) {
    if (y&1) res = (res+x)%mo;
    x = (x<<1)%mo;
    y >>= 1;
  }
  return res;
}
```

```
inline long long qmul(long long x, long long y, long long mo) {
  return (long long)((__int128)x*y%mo);
}
```

```
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;
}
```

57 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)
      ; }
};
```

58 快速傅里叶变换 |FFT

```
// array [0, n)
namespace FFT {
  static 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;
  }
}
```

59 快速数论变换 |NTT

```
// array [0, n)
namespace NTT {
  static 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;
  }
}
```

60 任意模数 NTT|MTT

```
namespace MTT {
  static const int SIZE = (1<<18)+7;
  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];
```

```cpp
    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];
    for (int i = 0; i < L; ++i) b[i] = dftb[i];
    fft(a, L), fft(b, L);
    for (int i = 0; i < L; ++i) {
      int da = (long long)(a[i].real / L + 0.5) % Mod;
      int db = (long long)(a[i].imag / L + 0.5) % Mod;
      int dc = (long long)(b[i].real / L + 0.5) % Mod;
      int dd = (long long)(b[i].imag / L + 0.5) % Mod;
      f[i] = (da + ((long long)(db + dc) << 15)
          + ((long long)dd << 30)) % Mod;
    }
    for (int i = 0; i < n+m-1; ++i) (f[i] += Mod) %= Mod;
  }
}
```

61 分治 FFT

```cpp
// give g[1, n) ask f[0, n)
// f[i] = sigma f[i-j]*g[j] (1 <= j <= i)
template <class T> // [l, r]
void cdq_fft(T f[], T g[], const int &l, const int &r) {
  if (r-l <= 1) return;
  int mid = (l+r)>>1;
  cdq_fft(f, g, l, mid);
  NTT::work(f+l, mid-l, g, r-l);
  for (int i = mid; i < r; ++i)
    (f[i] += NTT::f[i-l]) %= MOD;
  cdq_fft(f, g, mid, r);
}
```

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

62 快速沃尔什变换 |FWT

复杂度 $O(n\log n)|O(n2^n)$
$FWT(A \pm B) = FWT(A) \pm FWT(B)$
$FWT(cA) = cFWT(A)$
定义 $\bigoplus$ 为任意集合运算
$FWT(A \bigoplus B) = FWT(A) \times FWT(B)$
求 $C_i = \sum\limits_{i=j \bigoplus k} a_j b_k$

62.1 或运算

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

62.2 与运算

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

62.3 异或运算

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

62.4 code

```
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
```

63 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
```

63.1 倍增子集卷积

设多项式 $A=\sum\limits_{i=0}^{2^n-1}a_ix^i, B=\sum\limits_{i=0}^{2^n-1}b_ix^i$

求 $C=A*B=\sum\limits_{i=0}^{2^n-1}x^i\sum\limits_{d\subseteq i}a_db_{i-d}$

按照每个状态的最高位进行分组，然后卷 n 次

复杂度 $O(\sum\limits_{i=1}^{n}i^22^i)=O(n^22^n)$

```
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];
  }
}
```

64 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]);
}
```

65 约瑟夫环

$O(n)$

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

66 最小公倍数 lcm

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

67 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;
}
```

68 乘法逆元

68.1 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;
}
```

68.2 费马小定理

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

68.3 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;
}
```

69 中国剩余定理

69.1 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;
}
```

69.2 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;
}
```

70 排列组合奇偶性

C(n,k) 当 n&k == k 为奇数反之偶数

71 欧拉函数

```
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;
}
```

71.1 筛法

```
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;
```

72 莫比乌斯函数

```
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;
}
```

73 线性筛素数

```
struct Euler {
  int tot = 0;
  int prime[N];
  bool check[N];
  bool& operator [] (const int i) { return check[i]; }
  void init(int sz) {
    tot = 0;
    for (int i = 1; i <= sz; ++i) check[i] = true;
    check[1] = false;
    for (register int i = 2, j; i <= sz; ++i) {
      if (check[i]) prime[++tot] = i;
      for (j = 1; j <= tot && i*prime[j] <= sz; ++j) {
        check[i*prime[j]] = false;
        if (i%prime[j] == 0) break;
      }
    }
  }
} E;
```

74 判断素数 (质数)

74.1 Miller-Rabin 素性测试

```
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;
}
```

75 线性筛 GCD

```
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;
}
```

76 BSGS

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

```
// map<long long, int> mmp; // a^n = x
inline long long BSGS(long long a, long long x, long long m) {
  long long t = (long long)ceil(sqrt(m)); // b = a^i
  for(int i = 0; i < t; ++i)
    mmp[mul(x, qpow(a, i))] = i;
  a = qpow(a, t);
  long long now, ans; // now = (a^t)^i
  for(int i = 0; i <= t; ++i) {
    now = qpow(a, i);
    if(mmp.count(now)) {
      ans = t*i-mmp[now];
      if(ans > 0) return ans;
    }
  }
  return -1;
}
```

77 错排

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

78 原根

复杂度 $O(\sqrt{m} + g \times \log^2 m)$

```
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;
  }
}
```

Part VII
动态规划 DP

79 线性 DP

79.1 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])}
```

80 状压 DP

80.1 枚举子集

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

80.2 枚举 n 个元素大小为 k 的二进制子集

```
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~
}
```

81 背包问题

81.1 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]);
```

82 斜率优化

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

82.1 「HNOI2008」玩具装箱 TOY

$dp[i]=min(dp[j]+(sum[i]+i-sum[j]-j-L-1)^2)(j<i)$
令 $a[i]=sum[i]+i, b[i]=sum[i]+i+L+1$
$dp[i]=dp[j]+(a[i]-b[j])^2$
$dp[i]=dp[j]+a[i]^2-2\cdot a[i]\cdot b[j]+b[j]^2$
$2\cdot a[i]\cdot b[j]+dp[i]-a[i]^2=dp[j]+b[j]^2$
将 $b[j]$ 看作 $x, dp[j]+b[j]^2$ 看作 y，这个式子就可以看作一条斜率为 $2\cdot a[i]$ 的直线
而对于每个 i 来说，$a[i]$ 都是确定的，类似线性规划
$dp[i]$ 的含义转化为：当上述直线过点 $P(b[j], dp[j]+b[j]^2)$ 时，直线在 y 轴的截距加上 $a[i]^2$（一个定值）而题目即为找这个截距的最小值

83 四边形不等式

83.1 2D1D

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

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

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

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

83.2 1D1D

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

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

```
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)) i = k;
  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);
}
```

83.3 满足四边形不等式的函数类

- 性质 1 ：若函数 $w_1(l,r), w_2(l,r)$ 均满足四边形不等式（或区间包含单调性），则对于任意 $c_1,c_2 \geq 0$ ，函数 $c_1w_1+c_2w_2$ 也满足四边形不等式（或区间包含单调性）。
- 性质 2 ：若存在函数 $f(x),g(x)$ 使得 $w(l,r)=f(r)-g(l)$ ，则函数 w 满足四边形恒等式。当函数 f,g 单调增加时，函数 w 还满足区间包含单调性。
- 性质 3 ：设 $h(x)$ 是一个单调增加的凸函数，若函数 $w(l,r)$ 满足四边形不等式并且对区间包含关系具有单调性，则复合函数 $h(w(l,r))$ 也满足四边形不等式和区间包含单调性。
- 性质 4 ：设 $h(x)$ 是一个凸函数，若函数 $w(l,r)$ 满足四边形恒等式并且对区间包含关系具有单调性，则复合函数 $h(w(l,r))$ 也满足四边形不等式。

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

84 插头 DP| 轮廓线 DP

84.1 一个闭合回路

```
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];

```

```
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;
}
```

84.2 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;
        }
```

84.3 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
        s[j] = 0;
        int cnt = 0;
        for (int k = 1; k <= n; ++k) cnt += s[k] == b2;
        if (!b2 || cnt) insert(s, p.second);
        s[j] = b2;
        // choose
        if (!b1 && !b2) {
          s[j] = 7;
        } else {
          if (b1 > b2) swap(b1, b2); // in case b2 == 0
          s[j] = b2;
          if (b1) for (int k = 1; k <= n; ++k) if (s[k] == b1) s[k] =
              b2;
        }
        insert(s, p.second+a[i][j]);
      }
    }
  }
  cout << res << endl;
}
```

84.4 L 型

L 型地板：拐弯且仅拐弯一次。

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

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

Part VIII STL

85 unordered_map 重载

```
struct Node {
  int a, b;
  friend bool operator == (const Node &x, const Node &y) {
    return x.a == y.a && x.b == y.b;
  }
};
```

```cpp
// 方法一
namespace std {
  template <>
  struct hash<Node> {
    size_t operator () (const Node &x) const {
      return hash<int>()(x.a)^hash<int>()(x.b);
    }
  };
}
unordered_map<Node, int> mp;
// 方法二
struct KeyHasher {
  size_t operator () (const Node &x) const {
    return hash<int>()(x.a)^hash<int>()(x.b);
  }
};
unordered_map<Node, int, KeyHasher> mmp;
```