# Contents

# Chapter 1

# 几何

## 1.1 几何公式

- 三角形
  - 半周长
  $$P = \frac{a+b+c}{2}$$
  - 面积
  $$S = \frac{aH_a}{2} = \frac{ab\sin C}{2} = \sqrt{P(P-a)(P-b)(P-c)}$$
  - 中线
  $$M_a = \frac{\sqrt{2(b^2+c^2)-a^2}}{2} = \frac{\sqrt{b^2+c^2+2bc\cos A}}{2}$$
  - 角平分线
  $$T_a = \frac{\sqrt{bc((b+c)^2-a^2)}}{b+c} = 2\frac{bc\cos\frac{A}{2}}{b+c}$$
  - 高线
  $$H_a = b\sin C = c\sin B = \sqrt{b^2-(\frac{a^2+b^2-c^2}{2a})^2}$$
  - 内切圆半径
  $$r = \frac{S}{P} = a\frac{\sin\frac{B}{2}\sin\frac{C}{2}}{\sin\frac{B+C}{2}} = 4R\sin\frac{A}{2}\sin\frac{B}{2}\sin\frac{C}{2} = P\tan\frac{A}{2}\tan\frac{B}{2}\tan\frac{C}{2} = \sqrt{\frac{(P-a)(P-b)(P-c)}{P}}$$
  - 外接圆半径
  $$R = \frac{abc}{4S} = \frac{a}{2\sin A} = \frac{b}{2\sin B} = \frac{c}{2\sin C}$$
- 四边形
  $D_1, D_2$ 为对角线长，$M$ 为对角线中点连线，$A$ 为对角线夹角
  $$a^2+b^2+c^2+d^2 = D_1^2+D_2^2+4M^2$$
  $$S = D_1D_2\frac{\sin A}{2}$$
  以下对圆的内接四边形，$P$ 为半周长
  $$ac+bd = D_1D_2$$
  $$S = \sqrt{(P-a)(P-b)(P-c)(P-d)}$$
- 正 $n$ 边形
  $R$ 为外接圆半径，$r$ 为内切圆半径
  - 中心角
  $$A = \frac{2\pi}{n}$$
  - 内角
  $$C = \frac{(n-2)\pi}{n}$$
  - 边长
  $$a = 2\sqrt{R^2-r^2} = 2R\sin\frac{A}{2} = 2r\tan\frac{A}{2}$$
  - 面积
  $$S = \frac{nar}{2} = nr^2\tan\frac{A}{2} = \frac{nR^2\sin A}{2} = \frac{na^2}{4\tan\frac{A}{2}}$$
- 圆

- 弧长

$$l = rA$$

- 弦长

$$a = 2\sqrt{2hr - h^2} = 2r\sin\frac{A}{2}$$

- 弓形高

$$h = r - \sqrt{r^2 - \frac{a^2}{4}}$$
$$= r(1 - \cos\frac{A}{2}) = a\frac{\tan\frac{A}{4}}{2}$$

- 扇形面积

$$S_1 = \frac{rl}{2} = \frac{r^2A}{2}$$

- 弓形面积

$$S_2 = \frac{rl - a(r-h)}{2} = \frac{r^2(A - \sin A)}{2}$$

- 棱柱
  $A$ 为底面积，$h$ 为高，$l$ 为棱长，$p$ 为直截面周长
  - 体积

$$V = Ah$$

  - 侧面积

$$S = lp$$

  - 全面积

$$T = S + 2A$$

- 棱锥
  $A$ 为底面积，$h$ 为高
  - 体积

$$V = \frac{Ah}{3}$$

  以下对正棱锥，$l$ 为斜高，$p$ 为底面周长
  - 侧面积

$$S = \frac{lp}{2}$$

  - 全面积

$$T = S + A$$

- 棱台
  $A_1$，$A_2$ 为上下底面积，$h$ 为高
  - 体积

$$V = \frac{(A_1 + A_2 + \sqrt{A_1A_2})h}{3}$$

  以下为正棱台，$p_1$，$p_2$ 为上下底面周长，$l$ 为斜高
  - 侧面积

$$S = \frac{(p_1 + p_2)l}{2}$$

  - 全面积

$$T = S + A_1 + A_2$$

- 圆柱
  - 侧面积

$$S = 2\pi rh$$

  - 全面积

$$T = 2\pi r(h + r)$$

  - 体积

$$V = \pi r^2 h$$

- 圆锥
  - 母线

$$l = \sqrt{h^2 + r^2}$$

  - 侧面积

$$S = \pi rl$$

  - 全面积

$$T = \pi r(l + r)$$

  - 体积

$$V = \frac{\pi r^2 h}{3}$$

- 圆台
  - 母线

$$l = \sqrt{h^2 + (r_1 - r_2)^2}$$

  - 侧面积

$$S = \pi(r_1 + r_2)l$$

  - 全面积

$$T = \pi r_1(l + r_1) + \pi r_2(l + r_2)$$

  - 体积

$$V = \frac{\pi(r_1^2 + r_2^2 + r_1r_2)h}{3}$$

- 球
  - 全面积

$$T = 4\pi r^2$$

- 体积

$$V = \frac{4\pi r^3}{3}$$

- 球台
  - 侧面积

$$S = 2\pi rh$$

  - 全面积

$$T = \pi(2rh + r_1^2 + r_2^2)$$

  - 体积

$$V = \frac{\pi h(3(r_1^2 + r_2^2) + h^2)}{6}$$

- 球扇形
  $h$ 为球冠高，$r_0$ 为球冠底面半径
  - 全面积

$$T = \pi r(2h + r_0)$$

  - 体积

$$V = \frac{2\pi r^2 h}{3}$$

## 1.2 注意

1. 注意舍入方式 (0.5 的舍入方向)；防止输出 -0
2. 几何题注意多测试不对称数据；误差限缺省使用 1e-8
3. 整数几何注意 xmult 和 dmult 是否会出界；符点几何注意 eps 的使用
4. 避免使用斜率；注意除数是否会为 0
5. 公式一定要化简后再代入
6. 判断同一个 $2\pi$ 域内两角度差 (beta) 应该是
   abs(a1-a2) < beta || abs(a1-a2) > pi + pi - beta
   判断相等时将 beta 换成 eps
7. 需要的话尽量使用 atan2，注意：
   - atan2(0,0) = 0
   - atan2(1,0) = pi/2
   - atan2(-1,0) = -pi/2
   - atan2(0,1) = 0
   - atan2(0,-1) = pi
8. cross product = $|u||v|\sin\alpha$
   dot product = $|u||v|\cos\alpha$
9. $(P_1 - P_0) \times (P_2 - P0)$ 结果的意义：
   - 正：$< P_0, P_1 >$ 在 $< P_0, P_2 >$ 顺时针 $(0, \pi)$ 内
   - 负：$< P_0, P_1 >$ 在 $< P_0, P_2 >$ 逆时针 $(0, \pi)$ 内
   - 0 ：$< P_0, P_1 >$，$< P_0, P_2 >$ 共线，夹角为 $0$ 或 $\pi$

## 1.3 geo(猛犸也钻地)

```
// 计算几何 By 猛犸也钻地 @ 2012.08.21

/* 命名约定 //
   圆：圆心在 u，一般情况下半径 r 大于等于 0
   直线：经过点 u 和 v 的直线，u 不重合于 v
   射线：起点在 u，途经点 v，u 不重合于 v
   线段：起点在 u，终点在 v，u 不重合于 v
   散点集：点的可空集合
   多边形：至少有三个点，沿多边形的边依次排列，边不重合，图形不自交
   凸多边形：各内角均小于 180 度的多边形
   平面：由不共线的三点 uvw 所表示
```

```
// 所有函数都会默认传入的参数已满足上面的命名约定 */

#include <vector>
#include <cmath>
#include <utility>
#include <algorithm>
using namespace std;
using namespace rel_ops;

// typedef long long NUM;
typedef double NUM;
const NUM EPS = 1e-12, MAGIC = 2.71828e18;
// 因为有相对误差判断，所以 EPS 不要设得太宽

inline NUM sqr(NUM a) {
    return a*a;
}
inline NUM cmp(NUM a, NUM b) {
    // return a-b; // 坐标为浮点数时，使用下面这行
    return fabs(a-b)>=EPS+fabs(a)*EPS?a-b:0;
}

//-------------------------------------------------------------------//

struct VEC {
    NUM x,y;
} NOVEC = {MAGIC,MAGIC};
struct RAY {
    VEC u,v;
} NORAY = {NOVEC,NOVEC};
struct CIR {
    VEC u;
    NUM r;
} NOCIR = {NOVEC,MAGIC};

inline NUM sqr(const VEC &a) {
    return sqr(a.x)+sqr(a.y);
}
inline double abs(const VEC &a) {
    return sqrt(sqr(a));
}
inline NUM cmp(const VEC &a, const VEC &b) {
    NUM at=cmp(a.x,b.x);
    return !at?cmp(a.y,b.y):at;
}

inline VEC operator +(const VEC &a, const VEC &b) {
    return (VEC) {
        a.x+b.x,a.y+b.y
    };
}
inline VEC operator -(const VEC &a, const VEC &b) {
    return (VEC) {
        a.x-b.x,a.y-b.y
    };
```

```
}
inline NUM operator *(const VEC &a, const VEC &b) {
    return a.x*b.y-a.y*b.x;
}
inline NUM operator %(const VEC &a, const VEC &b) {
    return a.x*b.x+a.y*b.y;
}
inline VEC operator -(const VEC &a) {
    return (VEC) {
        -a.x,-a.y
    };
}
inline VEC operator ~(const VEC &a) {
    return (VEC) {
        -a.y,+a.x
    };
}
inline VEC operator *(NUM u, const VEC &a) {
    return (VEC) {
        u *a.x,u *a.y
    };
}
inline VEC operator *(const VEC &a, NUM u) {
    return (VEC) {
        a.x *u,a.y *u
    };
}
inline VEC operator /(const VEC &a, NUM u) {
    return (VEC) {
        a.x/u,a.y/u
    };
}
inline VEC operator /(const VEC &a, const VEC &b) {
    return a%b/sqr(b)*b;
}
inline bool operator ==(const VEC &a, const VEC &b) {
    return !cmp(a,b);
}
inline bool operator <(const VEC &a, const VEC &b) {
    return cmp(a,b)<0;
}

//  返回值        cmp_side             cmp_axis
//  ==  0       a 和 b 相互平行    /    a 和 b 相互垂直
//  <= -EPS     a 在 b 的左手侧    /    a 和 b 朝向相反（内角大于 90 度）
//  >= +EPS     a 在 b 的右手侧    /    a 和 b 朝向相同（内角小于 90 度）
NUM cmp_side(const VEC &a, const VEC &b) {
    return cmp(a.x*b.y,+a.y*b.x);
}
NUM cmp_axis(const VEC &a, const VEC &b) {
    return cmp(a.x*b.x,-a.y*b.y);
}

//-------------------------------------------------------------------//

```

```
// 求向量 a 长度缩放至 u 单位后的新向量，a 不能是零向量
// 求向量 a 绕坐标原点 o，逆时针转 u 度后的新向量
VEC resize(const VEC &a, NUM u) {
    return u/abs(a)*a;
}
VEC rotate(const VEC &a, NUM u) {
    return (VEC) {
        cos(u)*a.x-sin(u)*a.y,sin(u)*a.x+cos(u)*a.y
    };
}

// 点在直线上的投影（到直线的最近点）
// 点在圆周上的投影（到圆周的最近点）
VEC project(const VEC &p, const RAY &l) {
    return (p-l.u)/(l.v-l.u)+l.u;
}
VEC project(const VEC &p, const CIR &c) {
    if(!cmp(p,c.u)) return NOVEC;
    return resize(p-c.u,c.r)+c.u;
}

// 求两直线的交点
// 求直线与圆的交点，交线段的方向与原先直线相同
// 求两圆相交的交点，交线段的方向为圆心 a 到 b 连线方向逆指针转 90 度
// 求直线与凸多边形的交点，交线段的方向与原先直线相同，复杂度 O(logn)
VEC intersect(const RAY &a, const RAY &b) {
    VEC s=a.u-a.v,t=b.u-b.v;
    NUM at=cmp_side(s,t);
    if(!at) return NOVEC;
    return a.u+(b.u-a.u)*t/at*s;
}
RAY intersect(const RAY &l, const CIR &c) {
    VEC s=l.u+(c.u-l.u)/(l.v-l.u);
    NUM at=cmp(c.r*c.r,sqr(s-c.u));
    if(at<0) return NORAY;
    VEC t=resize(l.v-l.u,sqrt(at));
    return (RAY) {
        s-t,s+t
    };
}
RAY intersect(const CIR &a, const CIR &b) {
    NUM l=sqr(b.u-a.u);
    NUM w=(1+(a.r*a.r-b.r*b.r)/l)*0.5;
    NUM e=cmp(a.r*a.r/l,w*w);
    if(e<0) return NORAY;
    VEC t=sqrt(e)*~(b.u-a.u);
    VEC s=a.u+w*(b.u-a.u);
    return (RAY) {
        s-t,s+t
    };
}

// 判断三点是否共线
```

```
// 判断点在直线上的投影点，是否在线段上
// 判断点和线的位置关系，在外侧为 0，在直线上为 1 或 2(在线段上时为 2)
// 判断点和圆的位置关系，在外侧为 0，内部为 1，边上为 2
// 判断点和任意简单多边形的位置关系，在外侧为 0，内部为 1，边上为 2
// 快速地判断点和凸多边形的位置关系，在外侧为 0，内部为 1，边上为 2
// 判断两条线的位置关系，斜相交为 0，垂直为 1，平行为 2，重合为 3
bool collinear(const VEC &a, const VEC &b, const VEC &c) {
    return !cmp_side(a-b,b-c);
}
bool seg_range(const VEC &p, const RAY &l) {
    return cmp_axis(p-l.u,p-l.v)<=0;
}
int relation(const VEC &p, const RAY &l) {
    if(cmp_side(p-l.u,p-l.v)) return 0;
    return cmp_axis(p-l.u,p-l.v)>0?1:2;
}
int relation(const VEC &p, const CIR &c) {
    NUM at=cmp(sqr(c.r),sqr(c.u-p));
    return at?at<0?0:1:2;
}
int relation(const VEC &p, const vector<VEC> &u) {
    int n=u.size(),ret=0;
    for(int i=0; i<n; i++) {
        VEC s=u[i]-p,t=u[(i+1)%n]-p;
        if(t<s) swap(s,t);
        if(!cmp_side(s,t) && cmp_axis(s,t)<=0) return 2;
        if(cmp(s.x+p.x,p.x)<=0 && cmp(t.x+p.x,p.x)>0
                && cmp_side(s,t)>0) ret^=1;
    }
    return ret;
}
int relation_convex(const VEC &p, const vector<VEC> &u) {
    int n=u.size(),l=0,r=n-1,o=cmp_side(u[1]-u[0],u[r]-u[0])<0?-1:1;
    if(relation(p,(RAY) {
    u[0],u[1]
    })==2
    || relation(p,(RAY) {
        u[0],u[r]
    })==2) return 2;
    while(l<r) {
        int m=(l+r+1)/2;
        if(cmp_side(p-u[0],u[m]-u[0])*o<=0) l=m;
        else r=m-1;
    }
    if(!r || r==n-1) return 0;
    NUM at=cmp_side(p-u[r],u[r+1]-u[r])*o;
    return at?at<0:2;
}
int relation(const RAY &a, const RAY &b) {
    NUM at=cmp_side(a.u-a.v,b.u-b.v);
    return at?!cmp_axis(a.u-a.v,b.u-b.v):!cmp_side(a.u-b.u,a.u-b.v)+2;
}

// 由 ax+by+c=0 构造直线
```

```
// 由直径上的两点构造一个圆
// 由三角形的顶点构造外接圆
RAY make_line(NUM a, NUM b, NUM c) {
    if(!cmp(a,0) && !cmp(b,0)) return NORAY;
    else if(!cmp(a,0)) return (RAY) {
        {0,-c/b}, {1,-c/b}
    };
    else if(!cmp(b,0)) return (RAY) {
        {-c/a,0}, {-c/a,1}
    };
    return (RAY) {
        {0,-c/b}, {-c/a,0}
    };
}
CIR make_circle(const VEC &a, const VEC &b) {
    return (CIR) {
        (a+b)/2,abs(a-b)/2
    };
}
CIR make_circle(const VEC &a, const VEC &b, const VEC &c) {
    if(!cmp_side(a-b,a-c)) return NOCIR;
    NUM x=(c-b)%(a-c),y=(c-b)*(a-b);
    VEC m=(x/y*~(a-b)+a+b)/2;
    return (CIR) {
        m,abs(a-m)
    };
}

// 求三点的内切圆
// 求点到圆的两个切点，返回的切点分别在点到圆心连线方向的左侧和右侧
// 求两圆的两条公切线，切线段的方向与圆心 a 到 b 连线方向相同
//     默认是外公切线，若将其中的一个圆半径设为负数，则求出的是内公切线
CIR tangent_circle(const VEC &a, const VEC &b, const VEC &c) {
    if(!cmp_side(a-b,a-c)) return NOCIR;
    NUM x=abs(b-c),y=abs(c-a),z=abs(a-b);
    VEC m=(a*x+b*y+c*z)/(x+y+z);
    return (CIR) {
        m,fabs((m-a)*(a-b)*1.0/z)
    };
}
RAY tangent(const VEC &p, const CIR &c) {
    NUM l=sqr(p-c.u),e=cmp(l,c.r*c.r);
    if(e<0) return NORAY;
    NUM x=c.r/sqrt(l),y=sqrt(e/l);
    VEC s=resize(p-c.u,1),t=~s;
    RAY lr= {c.u+c.r *x *s-c.r *y*t,
             c.u+c.r *x *s+c.r *y*t
            };
    return lr;
}
pair<RAY,RAY> tangent(const CIR &a, const CIR &b) {
    NUM o=a.r-b.r,l=sqr(b.u-a.u),e=cmp(l,o*o);
    if(e<0) return make_pair(NORAY,NORAY);
    NUM x=o/sqrt(l),y=sqrt(e/l);
    VEC s=resize(b.u-a.u,1),t=~s;
```

```
    RAY ll= {a.u+a.r *x *s+a.r *y*t,
             b.u+b.r *x *s+b.r *y*t
            };
    RAY rr= {a.u+a.r *x *s-a.r *y*t,
             b.u+b.r *x *s-b.r *y*t
            };
    return make_pair(ll,rr);
}

// 由散点集构造一个最小覆盖圆, 期望复杂度 O(n)
CIR min_covering_circle(vector<VEC> u) {
    random_shuffle(u.begin(),u.end());
    int n=u.size(),i,j,k,z=1%n;
    CIR ret;
    for(ret=make_circle(u[0],u[z]),i=2; i<n; i++) if(!relation(u[i],ret))
            for(ret=make_circle(u[0],u[i]),j=1; j<i; j++) if(!relation(u[j],ret))
                    for(ret=make_circle(u[i],u[j]),k=0; k<j; k++) if(!relation(u[k],ret)\
)
                            ret=make_circle(u[i],u[j],u[k]);
    return ret;
}

// 求散点集的二维凸包, 并按逆时针顺序排列
// 若传入的点集不足以构成凸多边形, 则返回回的点集是退化后的点或线段
vector<VEC> convex_hull(vector<VEC> u) {
    sort(u.begin(),u.end()); // 这两行是排序 + 去重, 如果数据已经有保证
    u.erase(unique(u.begin(),u.end()),u.end()); // 则可省略相应的操作
    if(u.size()<3) return u;
    vector<VEC> c;
    for(size_t i=0,o=1,m=1; ~i; i+=o) {
        while(c.size()>m) {
            VEC a=c.back()-c[c.size()-2];
            VEC b=c.back()-u[i];
            if(cmp_side(a,b)<0) break; // 改成 <=0 则保留共线点
            c.pop_back();
        }
        c.push_back(u[i]);
        if(i+1==u.size()) m=c.size(),o=-1; // 条件成立时切换至上凸壳
    }
    c.pop_back();
    return c;
}

/* 警告: 下面这两个函数没有被正确地实现, 等待修正中
   比赛时请不要使用这两个函数, 有较高几率出错

// 求凸多边形上, 朝某个方向看过去的最远点的编号, 复杂度 O(logn)
// 如果有多解, 则返回相对于观测向量, 在凸多边形上相对顺序更靠前的点
int apoapsis(const VEC& v, const vector<VEC>& u){
    if(!cmp((VEC){0,0},v)) return -1;
```

```
    int l=0,r=u.size()-1;
    NUM s=cmp_axis(u[r]-u[0],v);
    NUM t=cmp_axis(u[1]-u[0],v);
    if(s<=0 && t<=0) return !s?r:0;
    while(l<r){
        int m=(l+r)/2,e=cmp_axis(u[m]-u[0],v);
        if((e>=0 && e<cmp_axis(u[m+1]-u[0],v))
        || (e< 0 && t<0)) l=m+1; else r=m;
    }
    return r;
}

// 求直线与凸多边形的交点，交线段的方向与原先直线相同，复杂度 O(logn)
RAY intersect(RAY l, const vector<VEC>& u){
    int n=u.size(),p,q,lo,hi;
    VEC o=l.v-l.u;
    if(cmp_side(u[1]-u[0],u[2]-u[0])<0) o=-o;
    NUM pt=cmp_side(o,u[p=apoapsis(~ o,u)]-l.u);
    NUM qt=cmp_side(o,u[q=apoapsis(~-o,u)]-l.u);
    if(pt*qt>0) return NORAY;
    for(;p<n+n;o=-o){ // 只执行两次，分别计算 (p,q] 和 (q,p] 段和直线的交点
        lo=p,hi=q+=n;
        swap(p+=n,q);
        while(lo<hi){
            int at=(lo+hi+1)/2;
            if(cmp_side(o,u[at%n]-l.u)>=0) lo=at; else hi=at-1;
        }
        if(!cmp_side(o,u[lo%n]-l.u)) l.u=u[lo%n];
        else l.u=intersect((RAY){u[lo%n],u[(lo+1)%n]},l);
        swap(l.u,l.v);
    }
    return l;
}

/*----------------------------------------------------------------------*/

struct TOR {
    NUM x,y,z;
} NOTOR = {MAGIC,MAGIC,MAGIC};
```

```
struct SIG {
    TOR u,v;
} NOSIG = {NOTOR,NOTOR};
struct PLN {
    TOR u,v,w;
} NOPLN = {NOTOR,NOTOR,NOTOR};

inline NUM sqr(const TOR &a) {
    return sqr(a.x)+sqr(a.y)+sqr(a.z);
}
inline double abs(const TOR &a) {
    return sqrt(sqr(a));
}
inline NUM cmp(const TOR &a, const TOR &b) {
    NUM at=cmp(a.x,b.x);
    if(!at) at=cmp(a.y,b.y);
    return !at?cmp(a.z,b.z):at;
}

inline TOR operator +(const TOR &a, const TOR &b) {
    return (TOR) {
        a.x+b.x,a.y+b.y,a.z+b.z
    };
}
inline TOR operator -(const TOR &a, const TOR &b) {
    return (TOR) {
        a.x-b.x,a.y-b.y,a.z-b.z
    };
}
inline TOR operator *(const TOR &a, const TOR &b) {
    return (TOR) {
        a.y *b.z-a.z *b.y,a.z *b.x-a.x *b.z,a.x *b.y-a.y *b.x
    };
}
inline NUM operator %(const TOR &a, const TOR &b) {
    return a.x*b.x+a.y*b.y+a.z*b.z;
}
inline TOR operator -(const TOR &a) {
    return (TOR) {
        -a.x,-a.y,-a.z
    };
}
inline TOR operator *(NUM u, const TOR &a) {
    return (TOR) {
        u *a.x,u *a.y,u *a.z
    };
}
inline TOR operator *(const TOR &a, NUM u) {
    return (TOR) {
        a.x *u,a.y *u,a.z *u
    };
}
inline TOR operator /(const TOR &a, NUM u) {
    return (TOR) {
        a.x/u,a.y/u,a.z/u
    };
}
inline TOR operator /(const TOR &a, const TOR &b) {
```

```
    return a%b/sqr(b)*b;
}
inline bool operator ==(const TOR &a, const TOR &b) {
    return !cmp(a,b);
}
inline bool operator <(const TOR &a, const TOR &b) {
    return cmp(a,b)<0;
}

// 下面两个函数类似于它们的二维版本，但 cmp_side 只能用于判定向量是否平行
int cmp_side(const TOR &a, const TOR &b) {
    return cmp(a.y*b.z,a.z*b.y)
           || cmp(a.z*b.x,a.x*b.z)
           || cmp(a.x*b.y,a.y*b.x);
}
NUM cmp_axis(const TOR &a, const TOR &b) {
    NUM x=a.x*b.x,y=a.y*b.y,z=a.z*b.z;
    if((x<0)==(y<0)) return cmp(x+y,-z);
    if((x<0)==(z<0)) return cmp(x+z,-y);
    // 注释掉上面两行可以提升程序速度，但有极小概率出现精度问题
    return cmp(y+z,-x);
}

//----------------------------------------------------------------------//

// 求平面 c 的法向量
// 求向量 a 长度缩放至 u 单位后的新向量，a 不能是零向量
// 求向量 a 绕转轴向量 o，逆时针转 u 度后的新向量
inline TOR normal(const PLN &c) {
    return (c.v-c.u)*(c.w-c.u);
}
TOR resize(const TOR &a, NUM u) {
    return u/abs(a)*a;
}
TOR rotate(const TOR &a, NUM u, const TOR &o) {
    return a*cos(u)+resize(o,1)*a*sin(u);
}

// 点在直线上的投影（到直线的最近点）
// 点在平面上的投影（到平面的最近点）
TOR project(const TOR &p, const SIG &l) {
    return (p-l.u)/(l.v-l.u)+l.u;
}
TOR project(const TOR &p, const PLN &c) {
    return (c.u-p)/normal(c)+p;
}

// 求两直线的交点
// 求直线与平面的交点
// 求两平面的交线
TOR intersect(const SIG &a, const SIG &b) {
```

```
    TOR s=b.u-b.v,p=s*(b.u-a.u);
    TOR t=a.u-a.v,q=s*t;
    if(cmp_axis(p,t) || !cmp_side(s,t)) return NOTOR;
    NUM at=cmp_axis(p,q);
    return a.u+(at?at<0?-1:1:0)*sqrt(sqr(p)/sqr(q))*t;
}
TOR intersect(const SIG &l, const PLN &c) {
    TOR at=l.v-l.u,o=normal(c);
    if(!cmp_axis(o,at)) return NOTOR;
    return l.u+(c.u-l.u)%o/(at%o)*at;
}
SIG intersect(const PLN &a, const PLN &b) {
    TOR o=normal(a);
    SIG s= {b.u,b.v},t= {b.u,b.w},r= {b.v,b.w};
    s.u=intersect(cmp_axis(s.u-s.v,o)?s:r,a);
    t.u=intersect(cmp_axis(t.u-t.v,o)?t:r,a);
    return (SIG) {
        s.u,t.u
    };
}

// 判断四点是否共面
// 判断三点是否共线
// 判断点在直线上的投影点，是否在线段上
// 判断点和线的位置关系，在外侧为 0，在直线上为 1 或 2(在线段上时为 2)
// 判断点和面的位置关系，在面内为 0，在正方向为 1，在负方向为 -1
// 判断两条线的位置关系，其他情况下为 0，垂直为 1，平行为 2，重合为 3
// 判断线和面的位置关系，斜相交为 0，垂直为 1，平行为 2，线在面内为 3
// 判断两平面的位置关系，斜相交为 0，垂直为 1，平行为 2，重合为 3
bool coplanar(const TOR &a, const TOR &b, const TOR &c, const TOR &d) {
    return !cmp_axis(a-b,(a-c)*(a-d));
}
bool collinear(const TOR &a, const TOR &b, const TOR &c) {
    return !cmp_side(a-b,b-c);
}
bool seg_range(const TOR &p, const SIG &l) {
    return cmp_axis(p-l.u,p-l.v)<=0;
}
int relation(const TOR &p, const SIG &l) {
    if(cmp_side(p-l.u,p-l.v)) return 0;
    return cmp_axis(p-l.u,p-l.v)>0?1:2;
}
int relation(const TOR &p, const PLN &c) {
    NUM at=cmp_axis(p-c.u,normal(c));
    return at?at<0?-1:1:0;
}
int relation(const SIG &a, const SIG &b) { // 注意，异面垂直也算垂直
    NUM at=cmp_side(a.u-a.v,b.u-b.v);
    return at?!cmp_axis(a.u-a.v,b.u-b.v):!cmp_side(a.u-b.u,a.u-b.v)+2;
}
int relation(const SIG &l, const PLN &c) {
    TOR o=normal(c),e=l.v-l.u;
```

```
    return cmp_axis(e,o)?!cmp_side(e,o):!cmp_axis(c.u-l.u,o)+2;
}
int relation(const PLN &a, const PLN &b) {
    TOR p=normal(a),q=normal(b);
    return cmp_side(p,q)?!cmp_axis(p,q):!cmp_axis(a.u-b.u,p)+2;
}

// 由 ax+by+cz+d=0 构造平面
PLN make_plane(NUM a, NUM b, NUM c, NUM d) {
    if(cmp(a,0)) return (PLN) {
        {-d/a,0,0}, {(-b-d)/a,1,0}, {(-c-d)/a,0,1}
    };
    if(cmp(b,0)) return (PLN) {
        {0,-d/b,0}, {1,(-a-d)/b,0}, {0,(-c-d)/b,1}
    };
    if(cmp(c,0)) return (PLN) {
        {0,0,-d/c}, {1,0,(-a-d)/c}, {0,1,(-b-d)/c}
    };
    return NOPLN;
}

// 求散点集的三维凸包，返回每个三角面的顶点编号，复杂度 O(nlogn)
// 从凸包外看，每个面的顶点都按逆时针的顺序排列，edge 存储了邻面编号
// 比如 edge::u[0] 表示的是：face::u[0] 至 face::u[1] 这条边所对应的邻面
// 传入的点集不能含有重点，若返回值为空集，则说明所有的点共面
// NUM 的类型为 long long 时，坐标的范围不要超过 10^6，建议使用浮点类型
struct TPL {
    int u[3];
};
vector<TPL> convex_hull(const vector<TOR> &p) {
    vector<TPL> face,edge;
    static vector<int> F[100005],G[100005*7]; // 注意设置最大结点数
    int n=p.size(),i,j,k;
    if(n<=3) return face;
    vector<int> u(n),v(4),at(n),go(n),by(n);
    for(i=0; i<n; i++) u[i]=i;
    random_shuffle(u.begin(),u.end());
    TOR a=p[u[0]]-p[u[1]],b;
    for(i=2; i<n; i++) if(cmp_side(a,b=p[u[0]]-p[u[i]])) break;
    for(j=i; j<n; j++) if(cmp_axis(a*b,p[u[0]]-p[u[j]])) break;
    if(i>=n || j>=n) return face;
    swap(u[i],u[2]),swap(u[j],u[3]);
    b=p[u[0]]+p[u[1]]+p[u[2]]+p[u[3]];
    for(i=0; i<4; i++) {
        a=(p[u[i]]-p[u[j=(i+1)%4]])*(p[u[i]]-p[u[k=(i+2)%4]]);
        if(cmp_axis(p[u[i]]*4-b,a)<0) swap(j,k),a=-a;
        face.push_back((TPL) {
            {
                u[i],u[j],u[k]
            }
        });
        edge.push_back((TPL) {
            {
```

```
                (k+1)%4,(i+1)%4,(j+1)%4
            }
        });
        for(j=4; j<n; j++) if(cmp_axis(p[u[j]]-p[u[i]],a)>0)
                F[j].push_back(i),G[i].push_back(j);
    }
    for(i=4; i<n; F[i++].clear()) {
        int x=n,m=F[i].size(),c=v.size();
        for(j=0; j<m; j++) v[F[i][j]]++;
        for(j=0; j<m; j++) if(v[F[i][j]]>0) {
                v[F[i][j]]=-1234567890;
                for(k=0; k<3; k++) {
                    if(v[edge[F[i][j]].u[k]]) continue;
                    at[x=face[F[i][j]].u[k]]=F[i][j];
                    go[x]=k;
                }
            }
        if(x==n) continue;
        for(j=x,k=-1; k!=x; j=k) {
            k=face[at[j]].u[(go[j]+1)%3];
            a=(p[j]-p[k])*(p[j]-p[u[i]]);
            int t=v.size(),w=edge[at[j]].u[go[j]];
            v.push_back(0);
            face.push_back((TPL) {
                {
                    j,k,u[i]
                }
            });
            edge.push_back((TPL) {
                {
                    w,t+1,t-1
                }
            });
            *find(edge[w].u,edge[w].u+3,at[j])=t;
            vector<int>::const_iterator o,z;
            z=set_union(G[at[j]].begin(),G[at[j]].end(),
                        G[w].begin(),G[w].end(),by.begin());
            for(o=by.begin(); o!=z; ++o)
                if(*o>i && cmp_axis(p[u[*o]]-p[u[i]],a)>0)
                    F[*o].push_back(t),G[t].push_back(*o);
        }
        edge[edge.back().u[1]=c].u[2]=edge.size()-1;
    }
    int m=v.size();
    for(i=j=0; i<m; G[i++].clear())
        if(!v[i]) face[j]=face[i],edge[j++]=edge[i];
    face.erase(face.begin()+j,face.end());
    edge.erase(edge.begin()+j,edge.end());
    return face;
}
```

## 1.4 geo

```
#include <cmath>
#include <algorithm>
using namespace std;
```

```cpp
const int MAXN = 1000;
const double eps = 1e-8, PI = atan2(0, -1);
inline double sqr(double x) {
    return x * x;
}
inline bool zero(double x) {
    return (x > 0 ? x : -x) < eps;
}
inline int sgn(double x) {
    return (x > eps ? 1 : (x + eps < 0 ? -1 : 0));
}
struct point {
    double x, y;
    point(double x, double y):x(x), y(y) {}
    point() {}
    bool operator == (const point &a) const {
        return sgn(x - a.x) == 0 && sgn(y - a.y) == 0;
    }
    bool operator != (const point &a) const {
        return sgn(x - a.x) != 0 || sgn(y - a.y) != 0;
    }
    bool operator < (const point &a) const {
        return sgn(x - a.x) < 0 || sgn(x - a.x) == 0 && sgn(y - a.y) < 0;
    }
    point operator + (const point &a) const {
        return point(x + a.x, y + a.y);
    }
    point operator - (const point &a) const {
        return point(x - a.x, y - a.y);
    }
    point operator * (const double &a) const {
        return point(x * a, y * a);
    }
    point operator / (const double &a) const {
        return point(x / a, y / a);
    }
    double operator * (const point &a) const {
        return x * a.y - y * a.x;     //xmult
    }
    double operator ^ (const point &a) const {
        return x * a.x + y * a.y;     //dmult
    }
    double length() const {
        return sqrt(sqr(x) + sqr(y));
    }
    point trunc(double a) const {
        return (*this) * (a / length());
    }
    point rotate(double ang) const {
        point p(sin(ang), cos(ang));
        return point((*this) * p, (*this) ^ p);
    }
    point rotate(const point &a) const {
        point p(-a.y, a.x);
        p = p.trunc(1.0);
        return point((*this) * p, (*this) ^ p);
    }
```

```
};
bool isConvex(int n, const point *p) {
    int i, s[3] = {1, 1, 1};
    for(i = 0; i < n && /*s[1] && */s[0] | s[2]; i++)
        s[sgn((p[(i + 1) % n] - p[i]) * (p[(i + 2) % n] - p[i])) + 1] = 0;
    return /*s[1] && */s[0] | s[2];
} //去掉注释即不允许相邻边共线
bool insideConvex(const point &q, int n, const point *p) {
    int i, s[3] = {1, 1, 1};
    for(i = 0; i < n && /*s[1] && */s[0] | s[2]; i++)
        s[sgn((p[(i + 1) % n] - p[i]) * (q - p[i])) + 1] = 0;
    return /*s[1] && */s[0] | s[2];
} //去掉注释即严格在形内
inline bool dotsInline(const point &p1, const point &p2, const point &p3) {
    return zero((p1 - p3) * (p2 - p3));
} //三点共线
inline int decideSide(const point &p1, const point &p2, const point &l1, const point &l2\
) {
    return sgn((l1 - l2) * (p1 - l2)) * sgn((l1 - l2) * (p2 - l2));
} //点 p1 和 p2, 直线 l1-l2,-1 表示在异侧,0 表示在线上,1 表示同侧
inline bool dotOnlineIn(const point &p, const point &l1, const point &l2) {
    return zero((p - l2) * (l1 - l2)) && (l1.x - p.x) * (l2.x - p.x) < eps && (l1.y - p.\
y) * (l2.y - p.y) < eps;
} //判点是否在线段及其端点上
inline bool parallel(const point &u1, const point &u2, const point &v1, const point &v2)\
 {
    return zero((u1 - u2) * (v1 - v2));
} //判直线平行
inline bool perpendicular(const point &u1, const point &u2, const point &v1, const point\
 &v2) {
    return zero((u1 - u2) ^ (v1 - v2));
} //判直线垂直
inline bool intersectIn(const point &u1, const point &u2, const point &v1, const point &\
v2) {
    if(!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2))
        return decideSide(u1, u2, v1, v2) != 1 && decideSide(v1, v2, u1, u2) != 1;
    else
        return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1, u1,\
 u2) || dotOnlineIn(v2, u1, u2);
} //判两线段相交，包括端点和部分重合
inline bool intersectEx(const point &u1, const point &u2, const point &v1, const point &\
v2) {
    return decideSide(u1, u2, v1, v2) < 0 && decideSide(v1, v2, u1, u2) < 0;
} //判两线段相交，不包括端点和部分重合
inline bool insidePolygon(const point &q, int n, const point *p, bool onEdge = true) {
    if(dotOnlineIn(q, p[n - 1], p[0])) return onEdge;
    for(int i = 0; i + 1 < n; i++) if(dotOnlineIn(q, p[i], p[i + 1])) return onEdge;
#define getq(i) Q[(sgn(p[i].x-q.x)>0)<<1|sgn(p[i].y-q.y)>0]
#define difq(a,b,i,j) (a==b?0:(a==((b+1)&3)?1:(a==((b+3)&3)?-1:(sgn((p[i]-q)*(p[j]-q))<<\
```

```
1))))
    int Q[4] = {2, 1, 3, 0}, oq = getq(n-1), nq = getq(0), qua = difq(nq, oq, n - 1, 0);
    oq = nq;
    for(int i = 1; i < n; i++) {
        nq = getq(i);
        qua += difq(nq, oq, i - 1, i);
        oq = nq;
    }
    return qua != 0; //象限环顾法, 较好
    /*point q1; int i = 0, cnt = 0; const double OFFSET = 1e6;//坐标上限
    for(q1 = point(rand() + OFFSET, rand() + OFFSET);i < n;) for(i = cnt = 0;i < n && !d\
otsInLine(q, q1, p[i]);i++) cnt += intersectEx(q, q1, p[i], p[(i + 1) % n]);
    return cnt & 1;*/ //考验 rp 的射线法
} //判点在任意多边形内
inline point intersection(const point &u1, const point &u2, const point &v1, const point\
 &v2) {
    return u1 + (u2 - u1) * (((u1 - v1) * (v1 - v2)) / ((u1 - u2) * (v1 - v2)));
} //求两直线交点, 须预判是否平行
inline point ptoline(const point &p, const point &l1, const point &l2) {
    point t = p;
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    return intersection(p, t, l1, l2);
} //点到直线的最近点, 注意 l1 不能等于 l2
inline double disptoline(const point &p, const point &l1, const point &l2) {
    return fabs((p - l2) * (l1 - l2)) / (l1 - l2).length();
} //点到直线距离, 注意 l1 不能等于 l2
inline point ptoseg(const point &p, const point &l1, const point &l2) {
    point t = p;
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    if(sgn((l1 - p) * (t - p)) * sgn((l2 - p) * (t - p)) > 0)
        return (p - l1).length() < (p - l2).length() ? l1 : l2;
    else
        return intersection(p, t, l1, l2);
} //点到线段的最近点, 注意 l1 不能等于 l2
inline double disptoseg(const point &p, const point &l1, const point &l2) {
    point t = point(l1.y - l2.y, l2.x - l1.x);
    if(sgn((l1 - p) * t) * sgn((l2 - p) * t) > 0)
        return min((p - l1).length(), (p - l2).length());
    else
        return disptoline(p, l1, l2);
} //点到线段距离, 注意 l1 不能等于 l2
double fermentpoint(int m, point p[]) {
    point u(0, 0), v;
    double step = 0, nowbest = 0, now, maxx = 0, maxy = 0;
    for(int i = 0; i < m; ++i) {
        u = u + p[i];
        maxx = max(maxx, fabs(p[i].x));
        maxy = max(maxy, fabs(p[i].y));
    }
```

```
    u = u / m;
    for(int i = 0; i < m; ++i) nowbest += (u - p[i]).length();
    for(step = maxx + maxy; step > 1e-10; step *= 0.97) //对结果有影响，注意调整
        for(int i = -1; i <= 1; i++)
            for(int j = -1; j <= 1; j++) {
                v = u + point(i, j) * step;
                now = 0;
                for(int i = 0; i < m; ++i) now += (v - p[i]).length();
                if(now < nowbest) {
                    nowbest = now;
                    u = v;
                }
            }
    return nowbest;
} //模拟退火求费马点
void polygonCut(int &n, point *p, const point &l1, const point &l2, const point &side) {
    int m = 0, i;
    point pp[MAXN]; //尽量定义成全局变量
    for(i = 0; i < n; i++) {
        if(decideSide(p[i], side, l1, l2) == 1) pp[m++] = p[i];
        if(decideSide(p[i], p[(i + 1) % n], l1, l2) < 1 && !(zero((p[i] - l2) * (l1 - l2\
)) && zero((p[(i + 1) % n] - l2) * (l1 - l2))))
            pp[m++] = intersection(p[i], p[(i + 1) % n], l1, l2);
    }
    for(n = i = 0; i < m; i++)
        if(!i || !zero(pp[i].x - pp[i - 1].x) || !zero(pp[i].y - pp[i-1].y)) p[n++] = pp\
[i];
    if(zero(p[n - 1].x - p[0].x) && zero(p[n - 1].y - p[0].y)) n--;
    if(n < 3) n = 0;
} //将多边形沿 l1,l2 确定的直线在 side 侧切割，保证 l1,l2,side 不共线
inline double Seg_area(const point &p1, const point &p2, const point &p0, double R) {
    point tmp = (p0 - p1).rotate(p2 - p1);
    double d = -tmp.y, h1 = -tmp.x, h2 = h1 + (p2 - p1).length();
    if(d >= R || d <= -R) return R * R * (atan2(d, h1) - atan2(d, h2));
    double dh = sqrt(R * R - d * d);
    if(h2 < -dh || dh < h1) return R * R * (atan2(d, h1) - atan2(d, h2));
    double ret = 0;
    if(h1 < -dh) ret += atan2(d, h1) - atan2(d, -dh);
    if(h2 > dh) ret += atan2(d, dh) - atan2(d, h2);
    return ret * R * R + d * (min(h2, dh) - max(h1, -dh));
} //圆与线段交的有向面积
int graham(int n, point *p, point *ch, bool comEdge = false) {
    if(n < 3) {
        for(int i = 0; i < n; i++) ch[i] = p[i];
        return n;
    }
    const double e1 = comEdge ? eps : -eps;
    int i, j, k;
    sort(p, p + n);
    ch[0] = p[0];
    ch[1] = p[1];
    for(i = j = 2; i < n; ch[j++] = p[i++]) while(j > 1 && (ch[j - 2] - ch[j - 1]) * (p[\
i] - ch[j - 1]) > e1) j--;
    ch[k = j++] = p[n - 2];
    for(i = n - 3; i > 0; ch[j++] = p[i--]) while(j > k && (ch[j - 2] - ch[j - 1]) * (p[\
```

```
i] - ch[j - 1]) > e1) j--;
    while (j > k && (ch[j - 2] - ch[j - 1]) * (ch[0] - ch[j - 1]) > e1) j--;
    return j;
} //求凸包,p 会被打乱顺序,ch 为逆时针,comEdge 为 true 时保留共线点, 重点会导致不稳定
```

## 1.5 geo3d

```
#include <cmath>
#include <algorithm>
using namespace std;
const int MAXN = 1000;
const double eps = 1e-8;
const double PI = atan2(0.0, -1.0);
inline double sqr(double x) {
    return x * x;
}
inline bool zero(double x) {
    return (x > 0 ? x : -x) < eps;
}
inline int sgn(double x) {
    return (x > eps ? 1 : (x + eps < 0 ? -1 : 0));
}
struct point3 {
    double x, y, z;
    point3(double x, double y, double z):x(x), y(y), z(z) {}
    point3() {}
    bool operator == (const point3 &a) const {
        return sgn(x - a.x) == 0 && sgn(y - a.y) == 0 && sgn(z - a.z) == 0;
    }
    bool operator != (const point3 &a) const {
        return sgn(x - a.x) != 0 || sgn(y - a.y) != 0 || sgn(z - a.z) != 0;
    }
    bool operator < (const point3 &a) const {
        return sgn(x - a.x) < 0 || sgn(x - a.x) == 0 && sgn(y - a.y) < 0 || sgn(x - a.x)\
 == 0 && sgn(y - a.y) == 0 && sgn(z - a.z) < 0;
    }
    point3 operator + (const point3 &a) const {
        return point3(x + a.x, y + a.y, z + a.z);
    }
    point3 operator - (const point3 &a) const {
        return point3(x - a.x, y - a.y, z - a.z);
    }
    point3 operator * (const double &a) const {
        return point3(x * a, y * a, z * a);
    }
    point3 operator / (const double &a) const {
        return point3(x / a, y / a, z / a);
    }
    point3 operator * (const point3 &a) const {
        return point3(y * a.z - z * a.y, z * a.x - x * a.z, x * a.y - y * a.x);    //xmu\
lt
    }
    double operator ^ (const point3 &a) const {
        return x * a.x + y * a.y + z * a.z;    //dmult
```

```
    }
    double sqrlen() const {
        return sqr(x) + sqr(y) + sqr(z);
    }
    double length() const {
        return sqrt(sqrlen());
    }
    point3 trunc(double a) const {
        return (*this) * (a / length());
    }
    point3 rotate(const point3 &a, const point3 &b, const point3 &c) const {
        return point3(a ^ (*this), b ^ (*this), c ^ (*this));    //abc正交且模为1
    }
};
inline point3 pvec(const point3 &a, const point3 &b, const point3 &c) {
    return (a - b) * (b - c);
} //平面法向量
inline bool dotsInline(const point3 &a, const point3 &b, const point3 &c) {
    return zero(((a - b) * (b - c)).length());
} //判三点共线
inline bool dotsOnplane(const point3 &a, const point3 &b, const point3 &c, const point3 \
&d) {
    return zero(pvec(a, b, c) ^ (d - a));
} //判四点共面
inline bool dotOnlineIn(const point3 &p, const point3 &l1, const point3 &l2) {
    return zero(((p - l1) * (p - l2)).length()) && (l1.x - p.x) * (l2.x - p.x) < eps && \
(l1.y - p.y) * (l2.y - p.y) < eps && (l1.z - p.z) * (l2.z - p.z) < eps;
} //判点是否在线段上，包括端点和共线
inline bool dotInplaneIn(const point3 &p, const point3 &a, const point3 &b, const point3\
 &c) {
    return zero(((a - b) * (a - c)).length() - ((p - a) * (p - b)).length() - ((p - b) *\
 (p - c)).length() - ((p - c) * (p - a)).length());
} //判点是否在空间三角形上，包括边界，须保证abc不共线
inline int decideSide(const point3 &p1, const point3 &p2, const point3 &l1, const point3\
 &l2) {
    return sgn(((l1 - l2) * (p1 - l2)) ^ ((l1 - l2) * (p2 - l2)));
} //点p1和p2，直线l1-l2,-1表示在异侧,0表示在线上,1表示同侧，须保证所有点共面
inline int decideSide(const point3 &p1, const point3 &p2, const point3 &a, const point3 \
&b, const point3 &c) {
    return sgn((pvec(a, b, c) ^ (p1 - a)) * (pvec(a, b, c) ^ (p2 - a)));
} //点p1和p2，平面abc,-1表示在异侧,0表示在面上,1表示同侧
inline bool parallel(const point3 &u1, const point3 &u2, const point3 &v1, const point3 \
&v2) {
    return zero(((u1 - u2) * (v1 - v2)).length());
} //判两直线平行
inline bool parallel(const point3 &a, const point3 &b, const point3 &c, const point3 &d,\
 const point3 &e, const point3 &f) {
    return zero((pvec(a, b, c) * pvec(d, e, f)).length());
} //判两平面平行
inline bool parallel(const point3 &l1, const point3 &l2, const point3 &a, const point3 &\
b, const point3 &c) {
```

```
    return zero((l1 - l2) ^ pvec(a, b, c));
} //判直线与平面平行
inline bool perpendicular(const point3 &u1, const point3 &u2, const point3 &v1, const po\
int3 &v2) {
    return zero((u1 - u2) ^ (v1 - v2));
} //判两直线垂直
inline bool perpendicular(const point3 &a, const point3 &b, const point3 &c, const point\
3 &d, const point3 &e, const point3 &f) {
    return zero(pvec(a, b, c) ^ pvec(d, e, f));
} //判两平面垂直
inline bool perpendicular(const point3 &l1, const point3 &l2, const point3 &a, const poi\
nt3 &b, const point3 &c) {
    return zero(((l1 - l2) * pvec(a, b, c)).length());
} //判直线与平面垂直
inline bool intersectIn(const point3 &u1, const point3 &u2, const point3 &v1, const poin\
t3 &v2) {
    if(!dotsOnplane(u1, u2, v1, v2)) return false;
    if(!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2))
        return decideSide(u1, u2, v1, v2) < 1 && decideSide(v1, v2, u1, u2) < 1;
    return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1, u1, u2)\
 || dotOnlineIn(v2, u1, u2);
} //判两线段相交，包括端点和部分重合
inline bool intersectEx(const point3 &u1, const point3 &u2, const point3 &v1, const poin\
t3 &v2) {
    return dotsOnplane(u1, u2, v1, v2) && decideSide(u1, u2, v1, v2) < 0 && decideSide(v\
1, v2, u1, u2) < 0;
} //判两线段相交，不包括端点和部分重合
inline bool intersect(const point3 &l1, const point3 &l2, const point3 &a, const point3 \
&b, const point3 &c, bool edge = true) {
    return decideSide(l1, l2, a, b, c) < edge && decideSide(a, b, l1, l2, c) < edge && d\
ecideSide(b, c, l1, l2, a) < edge && decideSide(c, a, l1, l2, b) < edge;
} //判线段与空间三角形相交,edge 表示是否包括交于边界和部分包含
point3 intersection(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &\
v2) {
    point3 p0 = (u1 - v1) * (v1 - v2), p1 = (u1 - u2) * (v1 - v2);
    return u1 + (u2 - u1) * (sgn(p0 ^ p1) * sqrt(p0.sqrlen() / p1.sqrlen()));
} //计算两直线交点，须预判直线是否共面和平行
point3 intersection(const point3 &l1, const point3 &l2, const point3 &a, const point3 &b\
, const point3 &c) {
    point3 temp = pvec(a, b, c);
    return l1 + (l2 - l1) * ((temp ^ (a - l1)) / (temp ^ (l2 - l1)));
} //计算直线与平面交点，须预判是否平行，并保证三点不共线
void intersection(const point3 &a, const point3 &b, const point3 &c, const point3 &d, co\
nst point3 &e, const point3 &f, point3 &p1, point3 &p2) {
    p1 = parallel(d, e, a, b, c) ? intersection(e, f, a, b, c) : intersection(d, e, a, b\
, c);
    p2 = parallel(f, d, a, b, c) ? intersection(e, f, a, b, c) : intersection(f, d, a, b\
, c);
} //计算两平面交线，注意事先判断是否平行，并保证三点不共线,p-q 为交线
inline double disptoline(const point3 &p, const point3 &l1, const point3 &l2) {
    return sqrt(((p - l1) * (l2 - l1)).sqrlen() / (l1 - l2).sqrlen());
```

```
} //点到直线距离
inline point3 ptoline(const point3 &p, const point3 &l1, const point3 &l2) {
    point3 temp = l2 - l1;
    return l1 + temp * ((p - l1) ^ temp) / (temp ^ temp);
} //点到直线最近点
inline double disptoplane(const point3 &p, const point3 &a, const point3 &b, const point\
3 &c) {
    point3 temp = pvec(a, b, c);
    return fabs(temp ^ (p - a)) / temp.length();
} //点到平面距离
inline point3 ptoplane(const point3 &p, const point3 &a, const point3 &b, const point3 &\
c) {
    return intersection(p, p + pvec(a, b, c), a, b, c);
} //点到平面最近点
inline double dislinetoline(const point3 &u1, const point3 &u2, const point3 &v1, const \
point3 &v2) {
    point3 temp = (u1 - u2) * (v1 - v2);
    return fabs((u1 - v1) ^ temp) / temp.length();
} //直线到直线距离
void linetoline(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &v2, \
point3 &p1, point3 &p2) {
    point3 ab = u2 - u1, cd = v2 - v1, ac = v1 - u1;
    p2 = v1 + cd * (((ab ^ cd) * (ac ^ ab) - (ab ^ ab) * (ac ^ cd)) / ((ab ^ ab) * (cd ^\
 cd) - sqr(ab ^ cd)));
    p1 = ptoline(p2, u1, u2);
} //直线到直线的最近点对,p1 在 u 上,p2 在 v 上, 须保证直线不平行
inline double angleCos(const point3 &u1, const point3 &u2, const point3 &v1, const point\
3 &v2) {
    return ((u1 - u2) ^ (v1 - v2)) / sqrt((u1 - u2).sqrlen() * (v1 - v2).sqrlen());
} //两直线夹角 cos 值
inline double angleCos(const point3 &a, const point3 &b, const point3 &c, const point3 &\
d, const point3 &e, const point3 &f) {
    point3 p1 = pvec(a, b, c), p2 = pvec(d, e, f);
    return (p1 ^ p2) / sqrt(p1.sqrlen() * p2.sqrlen());
} //两平面夹角 cos 值
inline double angleSin(const point3 &l1, const point3 &l2, const point3 &a, const point3\
 &b, const point3 &c) {
    point3 temp = pvec(a, b, c);
    return ((l1 - l2) ^ temp) / sqrt((l1 - l2).sqrlen() * temp.sqrlen());
} //直线平面夹角 sin 值
double angle(double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * PI / 180;
    while(dlng >= PI + PI) dlng -= PI + PI;
    if(dlng > PI) dlng = PI + PI - dlng;
    lat1 *= PI / 180;
    lat2 *= PI / 180;
    return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
} //计算大圆劣弧圆心角,lat(-90, 90) 表示纬度,lng 表示经度
```

## 1.6 三维几何

```cpp
//三维几何函数库
#include <cmath>

const double EPS = 1e-8;

struct Point3D {
    double x, y, z;
};

struct Line3D {
    Point3D a, b;
};

struct Plane {
    Point3D a, b, c;
};

struct PlaneF {
    // ax + by + cz + d = 0
    double a, b, c, d;
};

inline bool zero(double x) {
    return (x > 0 ? x : -x) < EPS;
}

//平方
inline double sqr(double d) {
    return d * d;
}

//计算 cross product U x V
inline Point3D xmult(const Point3D &u, const Point3D &v) {
    Point3D ret;
    ret.x = u.y * v.z - v.y * u.z;
    ret.y = u.z * v.x - u.x * v.z;
    ret.z = u.x * v.y - u.y * v.x;
    return ret;
}

//计算 dot product U . V
inline double dmult(const Point3D &u, const Point3D &v) {
    return u.x * v.x + u.y * v.y + u.z * v.z;
}

//矢量差 U - V
inline Point3D subt(const Point3D &u, const Point3D &v) {
    Point3D ret;
    ret.x = u.x - v.x;
    ret.y = u.y - v.y;
    ret.z = u.z - v.z;
    return ret;
```

```cpp
}

//取平面法向量
inline Point3D pvec(const Plane &s) {
    return xmult(subt(s.a, s.b), subt(s.b, s.c));
}
inline Point3D pvec(const Point3D &s1, const Point3D &s2, const Point3D &s3) {
    return xmult(subt(s1, s2), subt(s2, s3));
}
inline Point3D pvec(const PlaneF &p) {
    Point3D ret;
    ret.x = p.a;
    ret.y = p.b;
    ret.z = p.c;
    return ret;
}

//两点距离
inline double dis(const Point3D &p1, const Point3D &p2) {
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) + (p1.z - p2.z\
)*(p1.z - p2.z));
}

//向量大小
inline double vlen(const Point3D &p) {
    return sqrt(p.x*p.x + p.y*p.y + p.z*p.z);
}

//向量大小的平方
inline double sqrlen(const Point3D &p) {
    return (p.x*p.x + p.y*p.y + p.z*p.z);
}

//判三点共线
bool dotsInline(const Point3D &p1, const Point3D &p2, const Point3D &p3) {
    return sqrlen(xmult(subt(p1, p2), subt(p2, p3))) < EPS;
}

//判四点共面
bool dotsOnplane(const Point3D &a, const Point3D &b, const Point3D &c, const Point3D &d)\
 {
    return zero(dmult(pvec(a, b, c), subt(d, a)));
}

//判点是否在线段上，包括端点和共线
bool dotOnlineIn(const Point3D &p, const Line3D &l) {
    return zero(sqrlen(xmult(subt(p, l.a), subt(p, l.b)))) && (l.a.x - p.x) * (l.b.x - p\
.x) < EPS && (l.a.y - p.y) * (l.b.y - p.y) < EPS && (l.a.z - p.z) * (l.b.z - p.z) < EPS;
}
bool dotOnlineIn(const Point3D &p, const Point3D &l1, const Point3D &l2) {
    return zero(sqrlen(xmult(subt(p, l1), subt(p, l2)))) && (l1.x - p.x) * (l2.x - p.x) \
< EPS && (l1.y - p.y) * (l2.y - p.y) < EPS && (l1.z - p.z) * (l2.z - p.z) < EPS;
}
```

```
//判点是否在线段上，不包括端点
bool dotOnlineEx(const Point3D &p, const Line3D &l) {
    return dotOnlineIn(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y) || !zero(p.z -\
 l.a.z)) && (!zero(p.x - l.b.x) || !zero(p.y - l.b.y) || !zero(p.z - l.b.z));
}
bool dotOnlineEx(const Point3D &p, const Point3D &l1, const Point3D &l2) {
    return dotOnlineIn(p, l1, l2) && (!zero(p.x - l1.x) || !zero(p.y - l1.y) || !zero(p.\
z - l1.z)) && (!zero(p.x - l2.x) || !zero(p.y - l2.y) || !zero(p.z - l2.z));
}

//判点是否在空间三角形上，包括边界，三点共线无意义
bool dotInplaneIn(const Point3D &p, const Plane &s) {
    return zero(vlen(xmult(subt(s.a, s.b), subt(s.a, s.c))) - vlen(xmult(subt(p, s.a), s\
ubt(p, s.b))) - vlen(xmult(subt(p, s.b), subt(p, s.c))) - vlen(xmult(subt(p, s.c), subt(\
p, s.a))));
}
bool dotInplaneIn(const Point3D &p, const Point3D &s1, const Point3D &s2, const Point3D \
&s3) {
    return zero(vlen(xmult(subt(s1, s2), subt(s1, s3))) - vlen(xmult(subt(p, s1), subt(p\
, s2))) - vlen(xmult(subt(p, s2), subt(p, s3))) - vlen(xmult(subt(p, s3), subt(p, s1))))\
;
}

//判点是否在空间三角形上，不包括边界，三点共线无意义
bool dotInplaneEx(const Point3D &p, const Plane &s) {
    return dotInplaneIn(p, s) && sqrlen(xmult(subt(p, s.a), subt(p, s.b))) > EPS && sqrl\
en(xmult(subt(p, s.b), subt(p, s.c))) > EPS && sqrlen(xmult(subt(p, s.c), subt(p, s.a)))\
 > EPS;
}
bool dotInplaneEx(const Point3D &p, const Point3D &s1, const Point3D &s2, const Point3D \
&s3) {
    return dotInplaneIn(p, s1, s2, s3) && sqrlen(xmult(subt(p, s1), subt(p, s2))) > EPS \
&& sqrlen(xmult(subt(p, s2), subt(p, s3))) > EPS && sqrlen(xmult(subt(p, s3), subt(p, s1\
))) > EPS;
}

//判两点在线段同侧，点在线段上返回 0，不共面无意义
bool sameSide(const Point3D &p1, const Point3D &p2, const Line3D &l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p2, l.\
b))) > EPS;
}
bool sameSide(const Point3D &p1, const Point3D &p2, const Point3D &l1, const Point3D &l2\
) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2))) >\
 EPS;
}

//判两点在线段异侧，点在线段上返回 0，不共面无意义
bool oppositeSide(const Point3D &p1, const Point3D &p2, const Line3D &l) {
    return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p2, l.\
b))) < -EPS;
}
bool oppositeSide(const Point3D &p1, const Point3D &p2, const Point3D &l1, const Point3D\
```

```
&l2) {
    return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2))) <\
-EPS;
}

//判两点在平面同侧, 点在平面上返回 0
bool sameSide(const Point3D &p1, const Point3D &p2, const Plane &s) {
    return dmult(pvec(s), subt(p1, s.a)) * dmult(pvec(s), subt(p2, s.a)) > EPS;
}
bool sameSide(const Point3D &p1, const Point3D &p2, const Point3D &s1, const Point3D &s2\
, const Point3D &s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1)) * dmult(pvec(s1, s2, s3), subt(p2, s1))\
 > EPS;
}
bool sameSide(const Point3D &p1, const Point3D &p2, const PlaneF &s) {
    return (s.a * p1.x + s.b * p1.y + s.c * p1.z + s.d) * (s.a * p2.x + s.b * p2.y + s.c\
 * p2.z + s.d) > EPS;
}

//判两点在平面异侧, 点在平面上返回 0
bool oppositeSide(const Point3D &p1, const Point3D &p2, const Plane &s) {
    return dmult(pvec(s), subt(p1, s.a)) * dmult(pvec(s), subt(p2, s.a)) < -EPS;
}
bool oppositeSide(const Point3D &p1, const Point3D &p2, const Point3D &s1, const Point3D\
 &s2, const Point3D &s3) {
    return dmult(pvec(s1, s2, s3), subt(p1, s1)) * dmult(pvec(s1, s2, s3), subt(p2, s1))\
 < -EPS;
}
bool oppositeSide(const Point3D &p1, const Point3D &p2, const PlaneF &s) {
    return (s.a*p1.x+s.b*p1.y+s.c*p1.z+s.d) * (s.a*p2.x+s.b*p2.y+s.c*p2.z+s.d) < -EPS;
}

//判两直线平行
bool parallel(const Line3D &u, const Line3D &v) {
    return sqrlen(xmult(subt(u.a, u.b), subt(v.a, v.b))) < EPS;
}
bool parallel(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D &v2\
) {
    return sqrlen(xmult(subt(u1, u2), subt(v1, v2))) < EPS;
}

//判两平面平行
bool parallel(const Plane &u, const Plane &v) {
    return sqrlen(xmult(pvec(u), pvec(v))) < EPS;
}
bool parallel(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point3D &v1\
, const Point3D &v2, const Point3D &v3) {
    return sqrlen(xmult(pvec(u1, u2, u3), pvec(v1, v2, v3))) < EPS;
}
bool parallel(const PlaneF &u, const PlaneF &v) {
    return sqrlen(xmult(pvec(u), pvec(v))) < EPS;
}

//判直线与平面平行
bool parallel(const Line3D &l, const Plane &s) {
```

```
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}
bool parallel(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D &s2\
, const Point3D &s3) {
    return zero(dmult(subt(l1, l2), pvec(s1, s2, s3)));
}
bool parallel(const Line3D &l, const PlaneF &s) {
    return zero(dmult(subt(l.a, l.b), pvec(s)));
}

//判两直线垂直
bool perpendicular(const Line3D &u, const Line3D &v) {
    return zero(dmult(subt(u.a, u.b), subt(v.a, v.b)));
}
bool perpendicular(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3\
D &v2) {
    return zero(dmult(subt(u1, u2), subt(v1, v2)));
}

//判两平面垂直
bool perpendicular(const Plane &u, const Plane &v) {
    return zero(dmult(pvec(u), pvec(v)));
}
bool perpendicular(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point3\
D &v1, const Point3D &v2, const Point3D &v3) {
    return zero(dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)));
}
bool perpendicular(const PlaneF &u, const PlaneF &v) {
    return zero(dmult(pvec(u), pvec(v)));
}

//判直线与平面垂直
bool perpendicular(const Line3D &l, const Plane &s) {
    return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < EPS;
}
bool perpendicular(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3\
D &s2, const Point3D &s3) {
    return sqrlen(xmult(subt(l1, l2), pvec(s1, s2, s3))) < EPS;
}
bool perpendicular(const Line3D &l, const PlaneF &s) {
    return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < EPS;
}

//判两线段相交，包括端点和部分重合
bool intersectIn(const Line3D &u, const Line3D &v) {
    if (!dotsOnplane(u.a, u.b, v.a, v.b)) {
        return 0;
    } else if (!dotsInline(u.a, u.b, v.a) || !dotsInline(u.a, u.b, v.b)) {
        return !sameSide(u.a, u.b, v) && !sameSide(v.a, v.b, u);
    } else {
        return dotOnlineIn(u.a, v) || dotOnlineIn(u.b, v) || dotOnlineIn(v.a, u) || dotO\
nlineIn(v.b, u);
    }
}
bool intersectIn(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D \
```

```
&v2) {
    if (!dotsOnplane(u1, u2, v1, v2)) {
        return 0;
    } else if (!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2)) {
        return !sameSide(u1, u2, v1, v2) && !sameSide(v1, v2, u1, u2);
    } else {
        return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1, u1,\
 u2) || dotOnlineIn(v2, u1, u2);
    }
}

//判两线段相交，不包括端点和部分重合
bool intersectEx(const Line3D &u, const Line3D &v) {
    return dotsOnplane(u.a, u.b, v.a, v.b) && oppositeSide(u.a, u.b, v) && oppositeSide(\
v.a, v.b, u);
}
bool intersectEx(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D \
&v2) {
    return dotsOnplane(u1, u2, v1, v2) && oppositeSide(u1, u2, v1, v2) && oppositeSide(v\
1, v2, u1, u2);
}

//判线段与空间三角形相交，包括交于边界和（部分）包含
bool intersectIn(const Line3D &l, const Plane &s) {
    return !sameSide(l.a, l.b, s) && !sameSide(s.a, s.b, l.a, l.b, s.c) && !sameSide(s.b\
, s.c, l.a, l.b, s.a) && !sameSide(s.c, s.a, l.a, l.b, s.b);
}
bool intersectIn(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D \
&s2, const Point3D &s3) {
    return !sameSide(l1, l2, s1, s2, s3) && !sameSide(s1, s2, l1, l2, s3) && !sameSide(s\
2, s3, l1, l2, s1) && !sameSide(s3, s1, l1, l2, s2);
}

//判线段与空间三角形相交，不包括交于边界和（部分）包含
bool intersectEx(const Line3D &l, const Plane &s) {
    return oppositeSide(l.a, l.b, s) && oppositeSide(s.a, s.b, l.a, l.b, s.c) && opposit\
eSide(s.b, s.c, l.a, l.b, s.a) && oppositeSide(s.c, s.a, l.a, l.b, s.b);
}
bool intersectEx(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D \
&s2, const Point3D &s3) {
    return oppositeSide(l1, l2, s1, s2, s3) && oppositeSide(s1, s2, l1, l2, s3) && oppos\
iteSide(s2, s3, l1, l2, s1) && oppositeSide(s3, s1, l1, l2, s2);
}

//计算两直线交点，注意事先判断直线是否共面和平行！
//线段交点请另外判线段相交（同时还是要判断是否平行！）
#include <algorithm>

using namespace std;
Point3D intersection(Point3D u1, Point3D u2, Point3D v1, Point3D v2) {
    double dxu = u2.x - u1.x;
    double dyu = u2.y - u1.y;
    double dzu = u2.z - u1.z;
    double dxv = v2.x - v1.x;
    double dyv = v2.y - v1.y;
```

```
    double dzv = v2.z - v1.z;
    double t;
    if (!zero(dxu * dyv - dyu * dxv)) {
        t = (dyv * (v1.x - u1.x) + dxv * (u1.y - v1.y)) / (dxu * dyv - dyu * dxv);
    } else if (!zero(dxu * dzv - dzu * dxv)) {
        t = (dzv * (v1.x - u1.x) + dxv * (u1.z - v1.z)) / (dxu * dzv - dzu * dxv);
    } else {
        t = (dzv * (v1.y - u1.y) + dyv * (u1.z - v1.z)) / (dyu * dzv - dzu * dyv);
    }
    Point3D ret;
    ret.x = u1.x + dxu * t;
    ret.y = u1.y + dyu * t;
    ret.z = u1.z + dzu * t;
    return ret;
}

//计算直线与平面交点，注意事先判断是否平行，并保证三点不共线!
//线段和空间三角形交点请另外判断
Point3D intersection(const Line3D &l, const Plane &s) {
    Point3D ret = pvec(s);
    double t = (ret.x * (s.a.x - l.a.x) + ret.y * (s.a.y - l.a.y) + ret.z * (s.a.z - l.a\
.z)) / (ret.x * (l.b.x - l.a.x) + ret.y * (l.b.y - l.a.y) + ret.z * (l.b.z - l.a.z));
    ret.x = l.a.x + (l.b.x - l.a.x) * t;
    ret.y = l.a.y + (l.b.y - l.a.y) * t;
    ret.z = l.a.z + (l.b.z - l.a.z) * t;
    return ret;
}
Point3D intersection(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Poin\
t3D &s2, const Point3D &s3) {
    Point3D ret = pvec(s1, s2, s3);
    double t = (ret.x * (s1.x - l1.x) + ret.y * (s1.y - l1.y) + ret.z * (s1.z - l1.z)) /\
 (ret.x * (l2.x - l1.x) + ret.y * (l2.y - l1.y) + ret.z * (l2.z - l1.z));
    ret.x = l1.x + (l2.x - l1.x) * t;
    ret.y = l1.y + (l2.y - l1.y) * t;
    ret.z = l1.z + (l2.z - l1.z) * t;
    return ret;
}
Point3D intersection(const Line3D &l, const PlaneF &s) {
    Point3D ret = subt(l.b, l.a);
    double t = -(dmult(pvec(s), l.a) + s.d) / (dmult(pvec(s), ret));
    ret.x = ret.x * t + l.a.x;
    ret.y = ret.y * t + l.a.y;
    ret.z = ret.z * t + l.a.z;
    return ret;
}

//计算两平面交线，注意事先判断是否平行，并保证三点不共线!
Line3D intersection(const Plane &u, const Plane &v) {
    Line3D ret;
    ret.a = parallel(v.a, v.b, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.c) : \
intersection(v.a, v.b, u.a, u.b, u.c);
    ret.b = parallel(v.c, v.a, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.c) : \
intersection(v.c, v.a, u.a, u.b, u.c);
    return ret;
}
Line3D intersection(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point\
```

```
3D &v1, const Point3D &v2, const Point3D &v3) {
    Line3D ret;
    ret.a = parallel(v1, v2, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : intersecti\
on(v1, v2, u1, u2, u3);
    ret.b = parallel(v3, v1, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : intersecti\
on(v3, v1, u1, u2, u3);
    return ret;
}

//点到直线距离
double disptoline(const Point3D &p, const Line3D &l) {
    return vlen(xmult(subt(p, l.a), subt(l.b, l.a))) / dis(l.a, l.b);
}
double disptoline(const Point3D &p, const Point3D &l1, const Point3D &l2) {
    return vlen(xmult(subt(p, l1), subt(l2, l1))) / dis(l1, l2);
}

//点到直线最近点
Point3D ptoline(const Point3D &p, const Line3D &l) {
    Point3D ab = subt(l.b, l.a);
    double t = - dmult(subt(p, l.a), ab) / sqrlen(ab);
    ab.x *= t;
    ab.y *= t;
    ab.z *= t;
    return subt(l.a, ab);
}

//点到平面距离
double disptoplane(const Point3D &p, const Plane &s) {
    return fabs(dmult(pvec(s), subt(p, s.a))) / vlen(pvec(s));
}
double disptoplane(const Point3D &p, const Point3D &s1, const Point3D &s2, const Point3D\
 &s3) {
    return fabs(dmult(pvec(s1, s2, s3), subt(p, s1))) / vlen(pvec(s1, s2, s3));
}
double disptoplane(const Point3D &p, const PlaneF &s) {
    return fabs((dmult(pvec(s), p)+s.d) / vlen(pvec(s)));
}

//点到平面最近点
Point3D ptoplane(const Point3D &p, const PlaneF &s) {
    Line3D l;
    l.a = p;
    l.b = pvec(s);
    l.b.x += p.x;
    l.b.y += p.y;
    l.b.z += p.z;
    return intersection(l, s);
}

//直线到直线距离
double dislinetoline(const Line3D &u, const Line3D &v) {
    Point3D n = xmult(subt(u.a, u.b), subt(v.a, v.b));
    return fabs(dmult(subt(u.a, v.a), n)) / vlen(n);
}
```

```
double dislinetoline(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Poin\
t3D &v2) {
    Point3D n = xmult(subt(u1, u2), subt(v1, v2));
    return fabs(dmult(subt(u1, v1), n)) / vlen(n);
}

//直线到直线的最近点对
//p1 在 u 上，p2 在 v 上，p1 到 p2 是 uv 之间的最近距离
//注意，保证两直线不平行
void linetoline(const Line3D &u, const Line3D &v, Point3D &p1, Point3D &p2) {
    Point3D ab = subt(u.b, u.a), cd = subt(v.b, v.a), ac = subt(v.a, u.a);
    double r = (dmult(ab, cd) * dmult(ac, ab) - sqrlen(ab) * dmult(ac, cd)) / (sqrlen(ab\
)*sqrlen(cd) - sqr(dmult(ab, cd)));
    p2.x = v.a.x + r * cd.x;
    p2.y = v.a.y + r * cd.y;
    p2.z = v.a.z + r * cd.z;
    p1 = ptoline(p2, u);
}

//两直线夹角 cos 值
double angleCos(const Line3D &u, const Line3D &v) {
    return dmult(subt(u.a, u.b), subt(v.a, v.b)) / vlen(subt(u.a, u.b)) / vlen(subt(v.a,\
 v.b));
}
double angleCos(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D &\
v2) {
    return dmult(subt(u1, u2), subt(v1, v2)) / vlen(subt(u1, u2)) / vlen(subt(v1, v2));
}

//两平面夹角 cos 值
double angleCos(const Plane &u, const Plane &v) {
    return dmult(pvec(u), pvec(v)) / vlen(pvec(u)) / vlen(pvec(v));
}
double angleCos(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point3D &\
v1, const Point3D &v2, const Point3D &v3) {
    return dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)) / vlen(pvec(u1, u2, u3)) / vlen(pve\
c(v1, v2, v3));
}
double angleCos(const PlaneF &u, const PlaneF &v) {
    return dmult(pvec(u), pvec(v)) / (vlen(pvec(u)) * vlen(pvec(v)));
}

//直线平面夹角 sin 值
double angleSin(const Line3D &l, const Plane &s) {
    return dmult(subt(l.a, l.b), pvec(s)) / vlen(subt(l.a, l.b)) / vlen(pvec(s));
}
double angleSin(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D &\
s2, const Point3D &s3) {
    return dmult(subt(l1, l2), pvec(s1, s2, s3)) / vlen(subt(l1, l2)) / vlen(pvec(s1, s2\
, s3));
}
double angleSin(Line3D l, const PlaneF &s) {
    return dmult(subt(l.a, l.b), pvec(s)) / (vlen(subt(l.a, l.b)) * vlen(pvec(s)));
}
```

```cpp

// 平面方程形式转化 Plane -> PlaneF
PlaneF planeToPlaneF(const Plane &p) {
    PlaneF ret;
    Point3D m = xmult(subt(p.b, p.a), subt(p.c, p.a));
    ret.a = m.x;
    ret.b = m.y;
    ret.c = m.z;
    ret.d = -m.x * p.a.x - m.y * p.a.y - m.z * p.a.z;
    return ret;
}
```

## 1.7 三角形

```cpp
#include <cmath>

struct Point {
    double x, y;
};
struct Line {
    Point a, b;
};

inline double dis(const Point &p1, const Point &p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

Point intersection(const Line &u, const Line &v) {
    Point ret = u.a;
    double t = ((u.a.x - v.a.x) * (v.a.y - v.b.y) - (u.a.y - v.a.y) * (v.a.x - v.b.x)) /\
 ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
    ret.x += (u.b.x - u.a.x) * t;
    ret.y += (u.b.y - u.a.y) * t;
    return ret;
}

//外心
Point circumcenter(const Point &a, const Point &b, const Point &c) {
    Line u, v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
    return intersection(u, v);
}

//内心
Point incenter(const Point &a, const Point &b, const Point &c) {
    Line u, v;
    double m, n;
    u.a = a;
```

```
    m = atan2(b.y - a.y, b.x - a.x);
    n = atan2(c.y - a.y, c.x - a.x);
    u.b.x = u.a.x + cos((m + n) / 2);
    u.b.y = u.a.y + sin((m + n) / 2);
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x);
    n = atan2(c.y - b.y, c.x - b.x);
    v.b.x = v.a.x + cos((m + n) / 2);
    v.b.y = v.a.y + sin((m + n) / 2);
    return intersection(u, v);
}

//垂心
Point perpencenter(const Point &a, const Point &b, const Point &c) {
    Line u, v;
    u.a = c;
    u.b.x = u.a.x - a.y + b.y;
    u.b.y = u.a.y + a.x - b.x;
    v.a = b;
    v.b.x = v.a.x - a.y + c.y;
    v.b.y = v.a.y + a.x - c.x;
    return intersection(u, v);
}

//重心
//到三角形三顶点距离的平方和最小的点
//三角形内到三边距离之积最大的点
Point barycenter(const Point &a, const Point &b, const Point &c) {
    Line u, v;
    u.a.x = (a.x + b.x) / 2;
    u.a.y = (a.y + b.y) / 2;
    u.b = c;
    v.a.x = (a.x + c.x) / 2;
    v.a.y = (a.y + c.y) / 2;
    v.b = b;
    return intersection(u, v);
}
```

## 1.8 任意维空间最近点对

```
// 任意维最近点对
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;

const double EPS=1e-9, INF=1e9;
const int DIM=3; // 点集的维数

template <class T> T sqr(T x) {
    return x*x;
}

int fcmp(double x) {
```

```
    return fabs(x)<EPS ? 0 : (x<0 ? -1 : 1);
}

struct Point {
    double x[DIM];
    int id;
    double &operator[](int i) {
        return x[i];
    }
    friend double dis(Point a, Point b) {
        double res = 0;
        for(int i = 0; i < DIM; ++i)
            res += sqr(a[i] - b[i]);
        return sqrt(res);
    }
};

struct Comp {
    int dim;
    bool operator()(Point L, Point R) const {
        return L.x[dim] < R.x[dim];
    }
    Comp(int dim): dim(dim) {}
};

// 用 work() 函数找最近点对，从 min_dis 和 best_pair 中找到你需要的信息
// 复杂度约为 O(n*(log(n))^DIM)
struct ClosestPair {
    double min_dis; // 储存最近距离
    pii best_pair;  // 储存最近点对的标号
    vector<Point> vtmp[DIM];

    void init() {
        min_dis = INF;
        best_pair = pii(-1,-1);
        for(int i=0; i<DIM; ++i)
            vtmp[i].clear();
    }

    // 更新最近点对信息，这里是按照 (距离,a.id,b.id) 三元组作为比较标准
    // 请按实际需要修改
    void take_best(Point a, Point b) {
        if (a.id > b.id) swap(a, b);
        int cmp_res = fcmp(dis(a, b) - min_dis);
        if (
            best_pair.first==-1 || cmp_res<0 ||
            (cmp_res==0 && pii(a.id, b.id)<best_pair)
        ) {
            best_pair = pii(a.id, b.id);
            min_dis = dis(a, b);
        }
    }

    //在 v[l, r] 之间找出最近点对，请不要直接调用，而使用后面的 work() 函数
```

```
    void closest_pair(vector<Point> &v, int L, int R, int dim=0) {
        if (R-L <= 6) {
            for (int i=L+1; i<=R; ++i)
                for (int j=L; j<i; ++j)
                    take_best(v[i], v[j]);
            return;
        }

        if (dim+1 == DIM) {
            int z = dim - 1;
            for (int i=L; i<R; ++i) {
                for (int j=i+1; j<=R && v[j][z]-v[i][z]<min_dis; ++j)
                    take_best(v[i], v[j]);
            }
            return;
        }

        int M = (L+R) / 2;
        closest_pair(v, L, M, dim);
        closest_pair(v, M+1, R, dim);

        vector<Point> &u = vtmp[dim];
        u.clear();
        for(int i=L; i<=R; ++i)
            if (fcmp(abs(v[i][dim]-v[M][dim]) - min_dis) <= 0) {
                u.push_back(v[i]);
            }

        sort(u.begin(), u.end(), Comp(dim+1));
        if(!u.empty())closest_pair(u, 0, (int)u.size()-1, dim+1);
    }
public:
    double work(vector<Point> &v) {
        init();
        sort(v.begin(), v.end(), Comp(0));
        closest_pair(v, 0, (int)v.size()-1);
        return min_dis;
    }
} closest_pair;
```

## 1.9 圆

```
#include <cmath>

const double EPS = 1e-8;

struct Point {
    double x, y;
};

inline double xmult(const Point &p1, const Point &p2, const Point &p0) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
}

inline double dis(const Point &p1, const Point &p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}
```

```

double disptoline(const Point &p, const Point &l1, const Point &l2) {
    return fabs(xmult(p, l1, l2)) / dis(l1, l2);
}

Point intersection(const Point &u1, const Point &u2, const Point &v1, const Point &v2) {
    Point ret = u1;
    double t = ((u1.x - v1.x) * (v1.y - v2.y) - (u1.y - v1.y) * (v1.x - v2.x)) / ((u1.x \
- u2.x) * (v1.y - v2.y) - (u1.y - u2.y) * (v1.x - v2.x));
    ret.x += (u2.x - u1.x) * t;
    ret.y += (u2.y - u1.y) * t;
    return ret;
}

//判直线和圆相交，包括相切
int intersectLineCircle(const Point &c, double r, const Point &l1, const Point &l2) {
    return disptoline(c, l1, l2) < r + EPS;
}

//判线段和圆相交，包括端点和相切
int intersectSegCircle(const Point &c, double r, const Point &l1, const Point &l2) {
    double t1 = dis(c, l1) - r, t2 = dis(c, l2) - r;
    Point t = c;
    if (t1 < EPS || t2 < EPS) {
        return t1 > -EPS || t2 > -EPS;
    }
    t.x += l1.y - l2.y;
    t.y += l2.x - l1.x;
    return xmult(l1, c, t) * xmult(l2, c, t) < EPS && disptoline(c, l1, l2) - r < EPS;
}

//判圆和圆相交，包括相切
int intersectCircleCircle(const Point &c1, double r1, const Point &c2, double r2) {
    return dis(c1, c2) < r1 + r2 + EPS && dis(c1, c2) > fabs(r1 - r2) - EPS;
}

//计算圆上到点 p 最近点，如 p 与圆心重合，返回 p 本身
Point dotToCircle(const Point &c, double r, const Point &p) {
    Point u, v;
    if (dis(p, c)<EPS) {
        return p;
    }
    u.x = c.x + r * fabs(c.x - p.x) / dis(c, p);
    u.y = c.y + r * fabs(c.y - p.y) / dis(c, p) * ((c.x - p.x) * (c.y - p.y) < 0 ? -1 : \
1);
    v.x = c.x - r * fabs(c.x - p.x) / dis(c, p);
    v.y = c.y - r * fabs(c.y - p.y) / dis(c, p) * ((c.x - p.x) * (c.y - p.y) < 0 ? -1 : \
1);
    return dis(u, p) < dis(v, p) ? u : v;
}

//计算直线与圆的交点，保证直线与圆有交点
//计算线段与圆的交点可用这个函数后判点是否在线段上
void intersectionLineCircle(const Point &c, double r, const Point &l1, const Point &l2, \
```

```
Point &p1, Point &p2) {
    Point p = c;
    p.x += l1.y - l2.y;
    p.y += l2.x - l1.x;
    p = intersection(p, c, l1, l2);
    double t = sqrt(r * r - dis(p, c) * dis(p, c)) / dis(l1, l2);
    p1.x = p.x + (l2.x - l1.x) * t;
    p1.y = p.y + (l2.y - l1.y) * t;
    p2.x = p.x - (l2.x - l1.x) * t;
    p2.y = p.y - (l2.y - l1.y) * t;
}

//计算圆与圆的交点, 保证圆与圆有交点, 圆心不重合
void intersectionCircleCircle(const Point &c1, double r1, const Point &c2, double r2, Po\
int &p1, Point &p2) {
    Point u, v;
    double t = (1 + (r1 * r1 - r2 * r2) / dis(c1, c2) / dis(c1, c2)) / 2;
    u.x = c1.x + (c2.x - c1.x) * t;
    u.y = c1.y + (c2.y - c1.y) * t;
    v.x = u.x + c1.y - c2.y;
    v.y = u.y - c1.x + c2.x;
    intersectionLineCircle(c1, r1, u, v, p1, p2);
}
```

## 1.10 圆并

```
#include<bits/stdc++.h>
using namespace std;
//计算 n(n<MAXN) 个圆形面积的并, 并给出其质心坐标 (cx, cy)
//坐标和半径绝对值超过 10000 时需要修改避免爆 int
//用 add_circle(x, y, r) 添加圆
//多 case 时用 clear() 清空
//复杂度 $n^2 log n$
template<class T> T sqr(T x) {
    return x*x;
}
typedef double flt;
const flt EPS=1e-9;
const int MAXN=2001;
const flt PI=acos(-1.0);
struct CIRU {
    int x[MAXN],y[MAXN],r[MAXN],n;
    void clear() {
        n=0;
    }
    void add_circle(int xx,int yy,int rr) {
        x[n]=xx,y[n]=yy,r[n++]=rr;
    }
    int sqdis(int i, int j) {
        return sqr(x[i]-x[j])+sqr(y[i]-y[j]);
    }
    flt fix(flt x) {
        if(x<-PI)x+=2*PI;
```

```
        if(x>PI)x-=2*PI;
        return x;
    }
    pair<flt,int> v[MAXN*4+9];
    flt cx, cy;
    flt calc_area() {
        flt area=0;
        cx=cy=0;
        for(int i=0; i<n; ++i) {
            bool ok=1;
            int tot=2;
            v[0]=make_pair(-PI,0);
            v[1]=make_pair(PI,0);
            for(int j=0; j<n; ++j)if(j!=i) {
                    if(x[j]==x[i]&&y[j]==y[i]&&r[j]==r[i]&&j<i) {
                        ok=0;
                        break;
                    }
                    int dis=sqdis(i,j);
                    if(r[i]<r[j]&&dis<=sqr(r[i]-r[j])) {
                        ok=0;
                        break;
                    }
                    if(dis>=sqr(r[i]+r[j])||dis<=sqr(r[i]-r[j]))continue;
                    flt delta = (sqr(r[i])-sqr(r[j])+dis)*0.5;
                    delta = acos(delta / (sqrt(flt(dis))*r[i]));
                    flt dir = atan2(flt(y[j]-y[i]),flt(x[j]-x[i]));
                    flt bg=dir-delta, ed=dir+delta;
                    bg=fix(bg),ed=fix(ed);
                    v[tot++]=make_pair(bg,1);
                    v[tot++]=make_pair(ed,-1);
                    if(bg>ed+EPS) {
                        v[tot++]=make_pair(PI,-1);
                        v[tot++]=make_pair(-PI,1);
                    }
                }
            if(!ok)continue;
            if(tot==2) {
                flt s=2*PI*sqr(r[i]);
                area+=s; //与其他圆不相交
                cx+=s*x[i];
                cy+=s*y[i];
            } else {
                sort(v,v+tot);
                int cnt=0;
                for(int j=0; j+1<tot; ++j) {
                    cnt+=v[j].second;
                    flt a1=v[j].first,a2=v[j+1].first;
                    if(cnt==0 && a2-a1>EPS) {
                        //只算面积时可以只用下一行注释内的替换后面的所有
                        //area += r[i] * ((a2-a1)*r[i] + y[i]*(cos(a1)-cos(a2)) + x[i]*(\
sin(a2)-sin(a1)));
                        flt x1=x[i]+r[i]*cos(a1), y1=y[i]+r[i]*sin(a1);
                        flt x2=x[i]+r[i]*cos(a2), y2=y[i]+r[i]*sin(a2);
                        flt s_poly = x1*y2-x2*y1; //被包围的多边形面积
```

```
                        flt s_fan  = (a2-a1)*sqr(r[i]); //扇形面积
                        flt s_tri  = -sin(a2-a1)*sqr(r[i]); //扇形内的三角形面积，注意是负的
                        area+=s_poly+s_fan+s_tri;
                        flt sx=4.0/3.0*sin((a2-a1)/2)*r[i]*r[i]*r[i]/s_fan; //以下是求质心的
                        cx += (x1+x2)*s_poly/3;
                        cy += (y1+y2)*s_poly/3;
                        cx += (x[i]+sx*cos((a1+a2)/2))*s_fan;
                        cy += (y[i]+sx*sin((a1+a2)/2))*s_fan;
                        cx += (x1+x2+x[i])*s_tri/3;
                        cy += (y1+y2+y[i])*s_tri/3;
                    }
                }
            }
        }
        if(fabs(area)>EPS) {
            cx/=area;
            cy/=area;
        }
        return area*0.5;
    }
} cu;
```

## 1.11 球面

```
#include <cmath>

const double PI = acos(-1.0);

//计算圆心角 lat 表示纬度, - 90 <= w <= 90, lng 表示经度
//返回两点所在大圆劣弧对应圆心角, 0 <= angle <= PI
double angle(double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * PI / 180;
    while (dlng >= PI + PI) {
        dlng -= PI + PI;
    }
    if (dlng > PI) {
        dlng = PI + PI - dlng;
    }
    lat1 *= PI / 180;
    lat2 *= PI / 180;
    return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
}

//计算距离, r 为球半径
double lineDist(double r, double lng1, double lat1, double lng2, double lat2) {
    double dlng = fabs(lng1 - lng2) * PI / 180;
    while (dlng >= PI + PI) {
        dlng -= PI + PI;
    }
    if (dlng > PI) {
```

```
        dlng = PI + PI - dlng;
    }
    lat1 *= PI / 180;
    lat2 *= PI / 180;
    return r * sqrt(2 - 2 * (cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2)))\
;
}

//计算球面距离, r 为球半径
inline double sphereDist(double r, double lng1, double lat1, double lng2, double lat2) {
    return r * angle(lng1, lat1, lng2, lat2);
}
```

## 1.12 网格 (pick)

```
#include <cstdlib>

struct Point {
    int x, y;
};

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

//多边形上的网格点个数
int gridOnedge(int n, const Point *p) {
    int i, ret = 0;
    for (i = 0; i < n; i++) {
        ret += gcd(abs(p[i].x - p[(i + 1) % n].x), abs(p[i].y - p[(i + 1) % n].y));
    }
    return ret;
}

//多边形内的网格点个数
int gridInside(int n, const Point *p) {
    int i, ret = 0;
    for (i = 0; i < n; i++) {
        ret += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
    }
    return (abs(ret) - gridOnedge(n, p)) / 2 + 1;
}
```

# Chapter 2

# 组合

## 2.1 组合公式

```
1. C(m,n)=C(m,m-n)
2. C(m,n)=C(m-1,n)+C(m-1,n-1)

错排 (derangement)
D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)
     = (n-1)(D(n-2) + D(n-1))

Q(n) = D(n) + D(n-1)

Catalan numbers:
Ca(n)=C(2n-2,n-1)/n

K-dimensional Catalan numbers:
A(n) = 0! * 1! * .. * (k - 1)! * (k * n)! / (n! * (n + 1)! * .. * (n + k - 1)!)

2-d:
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900
3-d:
1, 5, 42, 462, 6006, 87516, 1385670, 23371634, 414315330
4-d:
1, 14, 462, 24024, 1662804, 140229804, 13672405890, 1489877926680

求和公式,k = 1..n
1. sum( k ) = n(n+1)/2
2. sum( 2k-1 ) = n^2
3. sum( k^2 ) = n(n+1)(2n+1)/6
4. sum( (2k-1)^2 ) = n(4n^2-1)/3
5. sum( k^3 ) = (n(n+1)/2)^2
6. sum( (2k-1)^3 ) = n^2(2n^2-1)
7. sum( k^4 ) = n(n+1)(2n+1)(3n^2+3n-1)/30
8. sum( k^5 ) = n^2(n+1)^2(2n^2+2n-1)/12
9. sum( k(k+1) ) = n(n+1)(n+2)/3
10. sum( k(k+1)(k+2) ) = n(n+1)(n+2)(n+3)/4
12. sum( k(k+1)(k+2)(k+3) ) = n(n+1)(n+2)(n+3)(n+4)/5
```

## 2.2 字典序全排列

```
//字典序全排列与序号的转换
int perm2Num(int n, const int *p) {
    int ret = 0, k = 1;
    for (int i = n - 2; i >= 0; k *= n - (i--)) {
        for (int j = i + 1; j < n; j++) {
            if (p[j] < p[i]) {
                ret += k;
            }
        }
    }
    return ret;
}

void num2Perm(int n, int *p, int t) {
    for (int i = n - 1; i >= 0; i--) {
        p[i] = t % (n - i);
        t /= n - i;
    }
    for (int i = n - 1; i; i--) {
        for (int j = i - 1; j >= 0; j--) {
            if (p[j] <= p[i]) {
                p[i]++;
            }
        }
    }
}
```

## 2.3 字典序组合

```
//字典序组合与序号的转换
//comb 为组合数 C(n, m), 必要时换成大数, 注意处理 C(n, m) = 0 | n < m
int comb(int n, int m) {
    int ret = 1;
    m = m < (n - m) ? m :(n - m);
    for (int i = n - m + 1; i <= n; ret *= (i++));
    for (int i = 1; i <= m; ret /= (i++));
    return m < 0 ? 0 : ret;
}

int comb2Num(int n, int m, const int *c) {
    int ret = comb(n, m);
    for (int i = 0; i < m; i++) {
        ret -= comb(n - c[i], m - i);
    }
    return ret;
}

void num2Comb(int n, int m, int *c, int t) {
    int j = 1, k;
    for (int i = 0; i < m; c[i++] = j++) {
        for (; t > (k = comb(n - j, m - i - 1)); t -= k, j++);
    }
}
```

## 2.4 排列组合生成

```cpp
//genPerm 产生字典序排列 P(n, m)
//genComb 产生字典序组合 C(n, m)
//genPermSwap 产生相邻位对换全排列 P(n, n)
//产生元素用 1..n 表示
//dummy 为产生后调用的函数, 传入 a[] 和 n, a[0]..a[n-1] 为一次产生的结果
const int MAXN = 100;

void dummy(const int *a, int n) {
    //...
}

void genPermRc(int *a, int n, int m, int k, int *temp, bool *tag) {
    if (k == m) {
        dummy(temp, m);
    } else {
        for (int i = 0; i < n; i++) {
            if (!tag[i]) {
                temp[k] = a[i], tag[i] = true;
                genPermRc(a, n, m, k + 1, temp, tag);
                tag[i] = 0;
            }
        }
    }
}

void genPerm(int n, int m) {
    int a[MAXN], temp[MAXN];
    bool tag[MAXN] = {false};
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
    }
    genPermRc(a, n, m, 0, temp, tag);
}

void genCombRc(int *a, int s, int e, int m, int &count, int *temp) {
    if (m == 0) {
        dummy(temp, count);
    } else {
        for (int i = s; i <= e - m + 1; i++) {
            temp[count++] = a[i];
            genCombRc(a, i + 1, e, m - 1, count, temp);
            count--;
        }
    }
}

void genComb(int n, int m) {
    int a[MAXN], temp[MAXN], count = 0;
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
    }
    genCombRc(a, 0, n-1, m, count, temp);
```

```
}

void genPermSwapRc(int *a, int n, int k, int *pos, int *dir) {
    int p1, p2, t;
    if (k == n) {
        dummy(a, n);
    } else {
        genPermSwapRc(a, n, k + 1, pos, dir);
        for (int i = 0; i < k; i++) {
            p2 = (p1 = pos[k]) + dir[k];
            t = a[p1];
            a[p1] = a[p2];
            a[p2] = t;
            pos[a[p1] - 1] = p1;
            pos[a[p2] - 1] = p2;
            genPermSwapRc(a, n, k + 1, pos, dir);
        }
        dir[k] = -dir[k];
    }
}

void genPermSwap(int n) {
    int a[MAXN], pos[MAXN], dir[MAXN];
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        pos[i] = i;
        dir[i] = -1;
    }
    genPermSwapRc(a, n, 0, pos, dir);
}
```

## 2.5 生成 gray 码

```
//生成 reflected gray code
//每次调用 gray 取得下一个码
//000...000 是第一个码,100...000 是最后一个码
void gray(int n, int *code) {
    int t = 0;
    for (int i = 0; i < n; t += code[i++]);
    if (t & 1) {
        for (n--; !code[n]; n--);
    }
    code[n - 1] = 1 - code[n - 1];
}
```

## 2.6 置换 (polya)

```
//求置换的循环节, polya 原理
//perm[0..n-1] 为 0..n-1 的一个置换 (排列)
//返回置换最小周期, num 返回循环节个数
const int MAXN = 1000;
```

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int polya(int *perm, int n, int &num) {
    int i, j, p, v[MAXN] = {0}, ret = 1;
    for (num = i = 0; i < n; i++) {
        if (!v[i]) {
            num++;
            p = i;
            for (j=0; !v[p = perm[p]]; j++) {
                v[p] = 1;
            }
            ret *= j / gcd(ret, j);
        }
    }
    return ret;
}
```

# Chapter 3

# 结构

## 3.1 ST 表

```
// RMQ
// MAXL = ceil(lg(MAXN))
// 根据需要重写以下函数

#define BIN(i) (1 << (i))
#define HLF(i) (BIN(i) >> 1)
#define PRE(i) ((i) > 0 ? (i) - 1 : 0)

// 为了省时间可以把 lg(x) 做成表，可以参考位运算加速
// 注意 lg(1 << i) = i - 1; lg((1 << i) + 1) = i;
const double eps = 1e-8;
const double ln2 = log(2.0);
inline int lg2(double x) {
    return (int)floor(fabs(log(x) / ln2 - eps));
}

/************ 一维 ************/

// MAXL = min{(1 << MAXL) >= MAXN};

template<int MAXL, class T = int, int MAXN = 1 << MAXL>
        struct RMQ {
            T e[MAXN];
            int rmq[MAXL][MAXN];

            // 重写 cmp，比较两个下标，返回较“小”下标
int cmp(int l, int r) {
    return e[l] <= e[r] ? l : r;
}

// 请直接对 e 赋值后调用
void init(const int n) {
    for (int i = 0; i < n; i++)
        rmq[0][i] = i;
```

```
    for (int i = 0; BIN(i + 1) <= n; i++)
        for (int j = 0; j <= n - BIN(i + 1); j++)
            rmq[i + 1][j] = cmp(rmq[i][j], rmq[i][j + BIN(i)]);
}

// [l, r) (l < r)
int index(int l, int r) {
    int b = lg2(r - l);
    return cmp(rmq[b][l], rmq[b][r - (1 << b)]);
}
T value(int l, int r) {
    return e[index(l, r)];
}
        };


/************ 二维 ************/

// 如果 MLE 就把 int 改成 short
typedef pair<int, int> IndexType;

// MAXR = min{(1 << MAXR) >= MAXM};
// MAXC = min((1 << MAXC) >= MAXN);

template<int MAXR, int MAXC, class T = int, int MAXM = 1 << MAXR, int MAXN = 1 << MAXC>
        struct RMQ2 {
            T e[MAXM][MAXN];
            IndexType rmq[MAXR][MAXC][MAXM][MAXN];

IndexType cmp(const IndexType &lhs, const IndexType &rhs) {
    return e[lhs.first][lhs.second] <= e[rhs.first][rhs.second] ? lhs : rhs;
}

void init(int m, int n) {
    for (int x = 0; x < m; x++)
        for (int y = 0; y < n; y++)
            rmq[0][0][x][y] = make_pair(x, y);
    for (int i = 0, ii; ii = PRE(i), BIN(i) <= m; i++)
        for (int j = 0, jj; jj = PRE(j), BIN(j) <= n; j++)
            for (int x = 0, xx; xx = HLF(i), x <= m - BIN(i); x++)
                for (int y = 0, yy; yy = HLF(j), y <= n - BIN(j); y++)
                    rmq[i][j][x][y] = cmp(
                        cmp(rmq[ii][jj][x]     [y], rmq[ii][jj][x]     [y + yy]),
                        cmp(rmq[ii][jj][x + xx][y], rmq[ii][jj][x + xx][y + yy])
                    );
}

IndexType index(int x1, int y1, int x2, int y2) {
    int xx = lg2(x2 - x1), yy = lg2(y2 - y1);
    return cmp(
        cmp(rmq[xx][yy][x1]            [y1], rmq[xx][yy][x1]            [y2 - (1 << yy)]\
),
        cmp(rmq[xx][yy][x2 - (1 << xx)][y1], rmq[xx][yy][x2 - (1 << xx)][y2 - (1 << yy)]\
)
    );
```

```
}

T value(int x1, int y1, int x2, int y2) {
    IndexType i = index(x1, y1, x2, y2);
    return e[i.first][i.second];
}
        };

/*
可以在开头定义一个全局变量，代替用浮点函数的 lg2

template<int MAXN>
struct LG2
{
    int lg2[MAXN + 1];

    LG2()
    {
        lg2[0] = -1;
        for (int i = 1; i <= MAXN; i++) {
            lg2[i] = lg2[i - 1] + ((i & (i - 1)) == 0);
        }
    }

    int operator()(int x) const { return lg2[x]; }
};

LG2<65536> lg2;
 */
```

## 3.2 Splay

```
#include <algorithm>
using namespace std;

/*
每次先调用一下 Splay::init()
不同的题目注意修改 newNode, pushdown, update 三个函数就足够了
Splay 命名空间里的函数都是对树的操作，对结点的操作都归类到 Node 里。
空间吃紧时可修改 erase 函数，增加内存池管理
*/
const int N = 130005;
struct Node {
```

```
    int key, size;
    bool rvs;
    Node *f, *ch[2];
    void set(int c, Node *x);      //设当前结点的左 (0)/右 (1) 儿子为 x
    void fix();                    //令两个儿子的父亲指针指向自己，主要为了写起来方便，意义明确
    void pushdown();               //标记下传
    void update();                 //从儿子处更新自己的信息
    void rotate();                 //向上旋转
    void Splay(Node *);            //把当前 Node 旋转到参数传入的 Node 下面。Node 默认为 null，直接调用 Splay() 则旋转到根
} statePool[N], *null;             //本模板统一用 null 代替 NULL
void Node::set(int c, Node *x) {
    ch[c] = x;
    x->f = this;
}
void Node::fix() {
    ch[0]->f = this;
    ch[1]->f = this;
}
void Node::pushdown() {
    if (this == null) return;
    if (rvs) {
        ch[0]->rvs ^= 1;
        ch[1]->rvs ^= 1;
        rvs = 0;
        swap(ch[0], ch[1]);
    }
}
void Node::update() {
    if (this == null) return;
    size = ch[0]->size + ch[1]->size + 1;
}
void Node::rotate() {
    Node *x = f;
    bool o = f->ch[0] == this;
    x->set(!o, ch[o]);
    x->f->set(x->f->ch[1] == x, this);
    set(o, x);
    x->update();
    update();
}
void Node::Splay(Node *goal = null) {
    pushdown();
    for (; f != goal;) {
        f->f->pushdown();
        f->pushdown();
        pushdown();
        if (f->f == goal) {
            rotate();
        } else if ((f->f->ch[0] == f) ^ (f->ch[0] == this)) {
            rotate();
```

```
            rotate();
        } else {
            f->rotate();
            rotate();
        }
    }
}
namespace Splay {
int poolCnt;
Node *newNode() {
    Node *p = &statePool[poolCnt++];
    p->f = p->ch[0] = p->ch[1] = null;
    p->size = 1;
    p->rvs = 0;
    return p;
}
//该命名空间里的函数必须先调用 init()
void init() {
    poolCnt = 0;
    null = newNode();
    null->size = 0;
}

//用 a[l..r] 的值建立一棵完全平衡的 Splay tree。返回树根。
template <class T> Node *build(int l, int r, T a[]) {
    if (l > r) return null;
    Node *p = newNode();
    int mid = (l + r) / 2;
    p->key = a[mid];
    if (l < r) {
        p->ch[0] = build(l, mid - 1, a);
        p->ch[1] = build(mid + 1, r, a);
        p->update();
        p->fix();
    }
    return p;
}

//返回树中第 i 个元素，若没有其它操作，请记得 select 后进行 Splay 操作以保证均摊复杂度。
Node *select(Node *root, int i) {
    for (Node *p = root;;) {
        p->pushdown();
        if (p->ch[0]->size + 1 == i) {
            return p;
        } else if (p->ch[0]->size >= i) {
            p = p->ch[0];
        } else {
            i -= p->ch[0]->size + 1;
            p = p->ch[1];
        }
    }
}

//返回结点 p 在树中的排名
int rank(Node *p) {
```

```
    p->Splay();
    return p->ch[0]->size + 1;
}

// 返回 >= a 的结点，若没有则返回 null，若之后没有其它操作，最好进行 Splay 操作以保证均摊复 \
杂度。
// 返回 null 时可以这样保证复杂度： select(root, root->size)->Splay();
template <class T> Node *lower_bound(Node *root, T a) {
    Node *ret = null;
    for (Node *p = root; p != null; ) {
        p->pushdown();
        if (a < p->key) {
            p = p->ch[1];
        } else {
            ret = p;
            p = p->ch[0];
        }
    }
    return ret;
}

//传入两树根，将之合并（可以处理 null）。p 中结点均在 q 中结点的左边。
Node *merge(Node *p, Node *q) {
    p->f = q->f = null;
    if (p == null) return q;
    if (q == null) return p;
    q = select(q, 1);
    q->Splay();
    q->set(0, p);
    q->update();
    return q;
}

//当 p 为根，且 q 为 p 的右儿子时，翻转从 p 到 q 的所有结点，返回树根
Node *reverse(Node *p, Node *q) {
    swap(p->ch[0], q->ch[0]);
    p->ch[1] = q->ch[1];
    q->ch[1] = p;
    q->f = null;
    p->fix();
    q->fix();
    p->update();
    q->update();
    p->ch[0]->rvs ^= 1;
    return q;
}

//翻转第 l 个元素到第 r 个元素，返回树根，下标从 1 开始。
Node *reverse(Node *root, int l, int r) {
    if (l >= r) return root;
    Node *p = select(root, l);
    p->Splay();
    Node *q = select(p, r);
    q->Splay(p);
```

```
    return reverse(p, q);
}

//在 p 的前一位插入 q, 返回树根
Node *insert(Node *p, Node *q) {
    p->Splay();
    if (p->ch[0] == null) {
        p->set(0, q);
    } else {
        Node *t = select(p, p->ch[0]->size);
        t->Splay(p);
        t->set(1, q);
        t->update();
    }
    p->update();
    return p;
}

//在第 i 个元素前插入结点 q, 返回树根
Node *insert(Node *root, Node *q, int i) {
    if (i > root->size) {
        Node *p = select(root, root->size);
        p->Splay();
        p->set(1, q);
        p->update();
        return p;
    } else {
        Node *p = select(root, i);
        return insert(p, q);
    }
}

//删除以 p 为根的子树
Node *erase(Node *p) {
    if (p->f != null) {
        Node *q = p->f;
        q->pushdown();
        q->set(q->ch[1] == p, null);
        q->update();
        q->Splay();
        return q;
    } else {
        return null;
    }
}

//删除第 l 个到第 r 个结点, 返回树根
Node *erase(Node *root, int l, int r) {
    if (l > r) return root;
    if (l == r) {
        Node *p = select(root, l);
        p->Splay();
        return merge(p->ch[0], p->ch[1]);
    } else {
        Node *p = select(root, l);
```

```
        p->Splay();
        Node *q = select(p, r);
        q->Splay(p);
        return merge(p->ch[0], q->ch[1]);
    }
}

//切开结点 p 与其左儿子之间的边，返回左子树的根以及原树的根。
pair <Node *, Node *> split(Node *p) {
    Node *q = p->ch[0];
    p->ch[0] = null;
    q->f = null;
    p->update();
    p->Splay();
    return make_pair(q, p);
}
}
```

## 3.3 划分树

```
// 划分树 by yxdb
// 功能：求任意一段区间第 K 小元素，也可求中位数. 仅能处理静态数组，可处理含有相同元素的情 \
况.
// 对于 N 个元素的数组，空间复杂度为 O(NLogN)，初始化时间复杂度为 O(NLogN)，每次查询时间复杂 \
度为 O(LogN).
// 查询前先调用 build(n, d) 初始化，d 数组 [0, n) 为需要处理的数据.
// query(l, r, k) 返回数组中 [l, r] 的第 k 小元素 (k 为 1 时即为求最小值).
// query2(l, r, x) 返回数组中小于 x 的元素个数 (仅在不含有相同元素的情况下测试过).
template <class T, int N>
class PartitionTree {
    pair<T,int> a[N];
    int p[25][N], L[25][N], R[25][N];
    bool isL[25][N];
    int sz;
    void build(int d, int l, int r) {
        if (l+1==r) return;
        int m=(l+r)>>1;
        for (int i=l, j=l, k=m; i<r; ++i) {
            L[d][i]=j, R[d][i]=k;
            p[d+1][(isL[d][i]=p[d][i]<m)?j++:k++]=p[d][i];
        }
        build(d+1, l, m);
        build(d+1, m, r);
    }
public:
    void build(int n, T *d) {
        sz = n;
        for (int i=0; i<n; ++i) a[i]=make_pair(d[i],i);
        sort(a, a+n);
        for (int i=0; i<n; ++i) p[0][a[i].second]=i;
        build(0, 0, n);
```

```
    }
    T query(int l, int r, int k) {
        int d, cnt;
        for (d=0; l<r; ++d) {
            cnt=L[d][r]-L[d][l]+isL[d][r];
            if (cnt>=k)  l=L[d][l], r=L[d][r]-!isL[d][r];
            else k-=cnt, l=R[d][l], r=R[d][r]- isL[d][r];
        }
        return a[p[d][l]].first;
        // 若改为 return a[p[d][l]].second; 可求第 k 小元素的位置.
    }
    int query2(int l, int r, T x) {
        int d, cnt, ret = 0, ll = 0, rr = sz, m, tmpl = l, tmpr = r;;
        for (d=0; l<r; ++d) {
            cnt=L[d][r]-L[d][l]+isL[d][r];
            if (a[m=ll+rr>>1].first > x) l=L[d][l], r=L[d][r]-!isL[d][r], rr=m;
            else ret += cnt, l=R[d][l], r=R[d][r]- isL[d][r], ll=m;
        }
        if (ret < tmpr-tmpl+1 && query(tmpl, tmpr, ret+1) < x) ret++;
        return ret;
    }
};
```

## 3.4 动态树

```
/*
    LCT 模板, 支持路径上的值的修改和查询。不支持对子树操作。

    除了 0 号结点, 其他节点都能使用, 清空时 memset f 数组就行了。

    函数中有//的地方是你有可能需要写代码的地方。

    维护的值必须都在点上, 如果值在边上, 可以把边看成额外的点。
    例如: link(边 id,x) link(边 id,y)

    struct sf 里面的 l,r,par,rev 是必须的信息, 如果需要维护其他的
    内容, 你可以自行添加。如果节点 x 的左子节点 (f[x].l) 或右子节点
    f[x].r) 是 0, 说明子节点不存在, 在使用下面说到的 push 和 update
    函数的时候要注意处理。

    如果自己使用不是模板里的函数对 splay 进行遍历或者其他操作时,
    不要忘记 push 和 update 函数的使用。

    常用的函数说明:

    push(x) 是将维护的信息下推给自己的左右儿子, 可以参考线段树。

```

```
    update(x) 是维护当前结点信息, 用例:
        f[x].mx=max(f[f[x].l].mx,f[f[x].r].mx);
        f[x].mx=max(f[x].mx,f[x].x);
        这样就维护了路径上最大值的信息。

    link(x,y) 是将 x 和 y 连接起来, 连接之前需要保证 x 和 y 属于不同的树。

    cut(x,y) 是将 (x,y) 这条边砍断, 砍断之前需要保证这条边存在。

    isconnect(x,y) 是检查 x 和 y 是否属于同一棵树。

    query(x,y) 是查询 x 到 y 这条路径上的信息, 用例:
        return f[y].mx;
        这样就返回了路径上的最大值。

    change(x,y) 是修改 x 到 y 这条路径上的值, 用例:
        f[y].xv+=v;
        这样就给路径上的值都加上了 v。

    不常用的函数说明:

    isroot(x) 是判断 x 节点是否为 splay 的根。

    rotate(x) 是把 x 节点旋转到 splay 上他的父亲节点的位置。

    splay(x) 是把 x 节点旋转到 splay 的根节点上。

    access(x) 是把 x 到 x 所在树的根路径上的所有节点组成一棵 splay,
    并且把 x 节点旋转到 splay 的根节点上。

    makeroot(x) 是把 x 节点变成他所在的树的根。
*/

#include<bits/stdc++.h>
#define MAXN 210000
using namespace std;

struct sf {
    int l,r,par,rev;
} f[MAXN];

bool isroot(int x) {
    return f[f[x].par].l!=x && f[f[x].par].r!=x;
}

```

```
void push(int x) {
    if (f[x].rev) {
        f[f[x].l].rev^=1;
        f[f[x].r].rev^=1;
        swap(f[x].l,f[x].r);
        f[x].rev=0;
    }
    //
}

void update(int x) {
    //
}

void rotate(int x) {
    int y=f[x].par,z=f[y].par;
    push(y);
    push(x);
    if (f[y].l==x) f[f[y].l=f[x].r].par=y,f[x].r=y;
    else f[f[y].r=f[x].l].par=y,f[x].l=y;
    if (f[z].l==y) f[z].l=x;
    if (f[z].r==y) f[z].r=x;
    f[f[y].par=x].par=z;
    update(y);
}

void splay(int x) {
    push(x);
    for (; !isroot(x); rotate(x)) {
        int y=f[x].par,z=f[y].par;
        if (!isroot(y)) rotate(f[z].l==y^f[y].l==x?x:y);
    }
    update(x);
}

void access(int x) {
    for(int t=0,y=x; y; y=f[y].par) splay(y),f[y].r=t,update(t=y);
    splay(x);
}

void makeroot(int x) {
    access(x);
    f[x].rev^=1;
}

void link(int x,int y) {
    makeroot(x);
    f[x].par=y;
}

void cut(int x,int y) {
    makeroot(x);
    access(y);
    f[x].par=f[y].l=0;
    update(y);
}

```

```
int query(int x,int y) {
    makeroot(x);
    access(y);
    //
}

void change(int x,int y) {
    makeroot(x);
    access(y);
    //
}

bool isconnect(int x,int y) {
    makeroot(x);
    access(y);
    while (push(y),f[y].l) y=f[y].l;
    splay(y);
    return x==y;
}

int main() {
}
```

## 3.5 子阵和

```
//求 sum{a[0..m-1][0..n-1]}
//维护和查询复杂度均为 O(logm*logn)
//用于动态求子阵和, 数组内容保存在 Sum.a[][] 中
//可以改成其他数据类型
#include <cstring>

const int MAXN = 10000;

inline int lowbit(int x) {
    return (x & -x);
}


template<class elemType>
class Sum {
    elemType a[MAXN][MAXN], c[MAXN][MAXN], ret;
    int m, n, t;
    void init(int i, int j) {
        memset(a, 0, sizeof(a));
        memset(c, 0, sizeof(c));
        m=i;
        n=j;
    }

    void update(int i, int j, elemType v) {
        for (v -= a[i][j], a[i++][j++] += v, t = j; i <= m; i += lowbit(i)) {
            for (j = t; j <= n; j += lowbit(j)) {
                c[i - 1][j - 1] += v;
```

```
            }
        }
    }

    elemType query(int i, int j) {
        for (ret = 0, t = j; i; i ^= lowbit(i)) {
            for (j = t; j; j ^= lowbit(j)) {
                ret += c[i - 1][j - 1];
            }
        }
        return ret;
    }
};
```

## 3.6 左偏树

```
// 左偏树是可以高效做合并操作的堆

#include <algorithm>    // swap
#include <functional>   // less

template<typename T = int, typename Pred = less<T> >
struct LeftistTree {
    struct node_type {
        T v;
        int d;
        node_type *l, *r;
        node_type(T v, int d) {
            this->v = v;
            this->d = d;
            l = NULL;
            r = NULL;
        }
        ~node_type() {
            delete l;
            delete r;
        }
    };
private:
    node_type *root;
    // 比较
    static Pred pr;
    // 核心操作，将以 l 和 r 为根的左偏树合并，返回新的根节点，复杂度 O(lgn)
    static node_type *merge(node_type *l, node_type *r) {
        if (l == NULL) {
            return r;
        }
        if (r == NULL) {
            return l;
        }

        if (pr(r->v, l->v)) {
            swap(l, r);
```

```
        }
        l->r = merge(l->r, r);
        if (l->r != NULL && (l->l == NULL || l->r->d > l->l->d)) {
            swap(l->l, l->r);
        }
        if (l->r == NULL) {
            l->d = 0;
        } else {
            l->d = l->r->d + 1;
        }

        return l;
    }
public:
    LeftistTree() {
        root = NULL;
    }
    ~LeftistTree() {
        delete root;
    }
    // 合并操作将让被合并的树变为空
    void merge(LeftistTree &t) {
        root = merge(root, t.root);
        t.root = NULL;
    }
    void push(T v) {
        root = merge(root, new node_type(v, 0));
    }
    void pop() {
        node_type *l = root->l, *r = root->r;

        root->l = NULL;
        root->r = NULL;
        delete root;
        root = merge(l, r);
    }
    T front() {
        return root->v;
    }
};
```

## 3.7 并查集

```
//带路径压缩的并查集，动态维护和查询等价类
//维护和查询复杂度略大于 O(1)
//集合元素取值 0 ~ MAXN-1,find(k) 返回所在的并查集根的编号
//init 默认不等价
const int MAXN = 1000000;
struct Dset {
    int p[MAXN];
    void init(int N) {
        for (int i = 0; i < N; ++i)
            p[i] = i;
    }
```

```
    int find(int k) {
        return p[k] == k ? k : (p[k] = find(p[k]));
    }
    void setFriend(int i, int j) {
        p[find(i)] = p[find(j)];
    }
    bool isFriend(int i, int j) {
        return find(i) == find(j);
    }
};
```

## 3.8 扩展并查集

```
//扩展并查集 by asmn
//已知许多组 v[i]^v[j]=k, 询问给某一对 v[i]^v[j] 的值
//setDiff(i,j,k) 输入 v[i]^v[j]=k 的关系
//返回值若为 false 表示输入的关系与之前有矛盾, 不予处理
//query(i,j,ans) 查询 v[i]^v[j] 的结果 ans
//返回若值为 false 表示结果无法由已有关系推得
#include <cstdio>
#include <algorithm>
using namespace std;
const int MAXN = 100000;
struct T {
    int prv, dif;
    T() {}
    T(int prv, int dif):prv(prv),dif(dif) {}
    T operator ^=(const T a) {
        return T(prv = a.prv, dif ^= a.dif);
    }
};
struct Dset {
    T p[MAXN];
    void init(int n) {
        for (int i = 0; i < n; ++i) {
            p[i] = T(i, 0);
        }
    }
    T find(int k) {
        return p[k].prv == k ? p[k] : (p[k] ^= find(p[k].prv));
    }
    // set v[i] ^ v[j] = k
    bool setDiff(int i, int j, int k) {
        T ti = find(i), tj = find(j);
        tj.dif ^= ti.dif ^ k;
        p[ti.prv] ^= tj;
        if (p[tj.prv].dif) {
            p[tj.prv].dif = 0;
            return false;
        } else {
            return true;
        }
```

```
    }
    //query ans = v[i] ^ v[j]
    bool query(int i, int j, int &ans) {
        ans = find(i).dif ^ find(j).dif;
        return (find(i).prv == find(j).prv);
    }
};
int main() {
    int i, j, ans;
    char op;
    Dset A;
    A.init(100);
    while (scanf(" %c %d%d", &op, &i, &j) != EOF) {
        switch (op) {
        case 'f':
            if (A.setDiff(i, j, 0)) {
                puts("OK");
            } else {
                puts("NO");
            }
            break;
        case 'e':
            if (A.setDiff(i, j, 1)) {
                puts("OK");
            } else {
                puts("NO");
            }
            break;
        case 'q':
            if (A.query(i, j, ans)) {
                printf("%s\n", ans ? "e" : "f");
            } else {
                puts("I dont know");
            }

        }
    }
}
```

## 3.9 树状数组

```
// 树状数组 By 猛犸也钻地 @ 2011.11.24

/* 使用提示 //
   单点修改/区间查询:
       使 P 位置增加 V           modify(P,V);
       查询 [L,R] 区间的和      getsum(R)-getsum(L-1);
   区间修改/单点查询:
       使 [L,R] 区间各增加 V     modify(L,V),modify(R+1,-V);
       查询 P 位置的值          getsum(P);
   区间修改/区间查询:
```

```
        使 [0,R] 区间各增加 V     A.modify(R,R*V),B.modify(R,-V);
        查询 [0,R] 区间的和      A.getsum(R)+B.getsum(R)*R;
    多维树状数组和单维的类似, 比如修改操作是这样的:
        for(int i=x+BIAS;i<SIZE;i++)
            for(int j=y+BIAS;j<SIZE;j++) u[i][j]+=v;
    模板里的 BIAS 用来避免下溢出, 比如查询 L-1 时, 可能会取到 0 或 -1 的位置
// 因此 SIZE 通常设为最大结点数 +2*BIAS, 而 BIAS 通常设为 5 或 10 */

#include <cstring>
using namespace std;

class BITree {
public:
    static const int SIZE = 100010, BIAS = 5;
    long long u[SIZE];
    void clear() {
        memset(u,0,sizeof(u));
    }
    void modify(int x, long long v) {
        for(x+=BIAS; x<SIZE; x+=x&-x) u[x]+=v;
    }
    long long getsum(int x) {
        long long s=0;
        for(x+=BIAS; x; x-=x&-x) s+=u[x];
        return s;
    }
};
```

## 3.10 树链剖分

```
// 树链剖分 By 猛犸也钻地 @ 2012.02.10

#include <vector>
#include <cstring>
using namespace std;

class TreeDiv {
public:
    static const int SIZE = 100005; // SIZE 为最大结点数 +1
    int sz[SIZE],lv[SIZE]; // sz[] 为子树的总结点数, lv[] 为结点的深度
    int rt[SIZE],fa[SIZE]; // rt[] 为结点所在的树根, fa[] 为结点的父亲
    // seg[] 为结点所在的链的编号, idx[] 为结点在链上的位置
    // low[] 为链首结点, top[] 为链尾结点的父亲, len[] 为链上结点的个数
    int cnt,seg[SIZE],idx[SIZE],low[SIZE],top[SIZE],len[SIZE];
    // 遍历某条链的方法: for(int x=low[u];x!=top[u];x=fa[x])
    // 传入结点个数 n 及各结点的出边 e[], 对树或森林进行剖分, 返回链数 cnt
    int init(int n, const vector<int> e[]) {
        memset(len,cnt=0,sizeof(len));
```

```
        memset(lv,0,sizeof(lv));
        for(int i=0; i<n; i++) if(!lv[i]) {
                segment(rt[i]=fa[i]=i,e);
                top[seg[i]]=fa[i]=SIZE-1; // 指向虚根
            }
        return cnt;
    }
    // 求 x 和 y 的最近公共祖先，不在同一棵树上则返回 -1，复杂度 O(logN)
    int lca(int x, int y) {
        if(rt[x]!=rt[y]) return -1;
        while(seg[x]!=seg[y]) {
            int p=top[seg[x]],q=top[seg[y]];
            if(lv[p]>lv[q]) x=p;
            else y=q;
        }
        return lv[x]<lv[y]?x:y;
    }
private:
    void segment(int x, const vector<int> e[]) {
        int y,t=-1;
        sz[x]=1;
        rt[x]=rt[fa[x]];
        lv[x]=lv[fa[x]]+1;
        for(size_t i=0; i<e[x].size(); i++) if(e[x][i]!=fa[x]) {
                fa[y=e[x][i]]=x;
                segment(y,e);
                sz[x]+=sz[y];
                if(t<0 || sz[y]>sz[t]) t=y;
            }
        seg[x]=~t?seg[t]:cnt;
        idx[x]=~t?idx[t]+1:0;
        if(t<0) low[cnt++]=x;
        len[seg[x]]++;
        top[seg[x]]=fa[x];
    }
};
```

## 3.11 矩形并

```
//线段树求矩形并得面积和周长
//要保证传入的 x1 < x2, y1 < y2
#include<algorithm>
#include <cstring>
#define MAXN 40000
using namespace std;
typedef long long LL;
const int K = 1;
int tot, X[MAXN * 2], CX;
pair<pair<int, int>, pair<int,int> > E[MAXN * 2];

//矩形类
struct Rect {
    int x1, y1, x2, y2;
```

```
} R[MAXN];

struct SegTree {
    int L, R, Lson, Rson, cover, length[K + 1];
    void build(int, int);
    void add(int, int, int);
} T, A[MAXN * 4];

void SegTree::build(int l, int r) {
    L = l;
    R = r;
    cover = 0;
    memset(length, 0, sizeof(length));
    length[0] = X[R + 1] - X[L];
    if (L == R) return ;
    int mid = (L + R) >> 1;
    A[Lson = tot++].build(l, mid);
    A[Rson = tot++].build(mid + 1, r);
}
//线段树的填删线段，只能删除之前加入过的线段
void SegTree::add(int v, int l, int r) {
    memset(length, 0, sizeof(length));
    if (l <= L && R <= r) {
        cover += v;
        if (L == R) {
            length[min(cover, K)] += X[R + 1] - X[L];
        } else {
            for (int i = 0; i <= K; ++i) {
                length[min(i + cover, K)] += A[Lson].length[i] + A[Rson].length[i];
            }
        }
        return ;
    }
    int mid = (L + R) >> 1;
    if (l <= mid) A[Lson].add(v, l, r);
    if (r > mid) A[Rson].add(v, l, r);
    for (int i = 0; i <= K; ++i) {
        length[min(i + cover, K)] += A[Lson].length[i] + A[Rson].length[i];
    }
}

//此子函数用于把坐标离散化至 X 数组，建立线段树，并把边的事件用 y 坐标排序
void discrete(Rect *R, int N) {
    int i, tx1, tx2;
    for (CX = i = 0; i < N; ++i) {
        X[CX++] = R[i].x1;
        X[CX++] = R[i].x2;
    }
    sort(X, X + CX);
    CX = unique(X, X + CX) - X;
    T.build(tot = 0, CX - 2);
    for (i = 0; i < N; ++i) {
        tx1 = lower_bound(X, X + CX, R[i].x1) - X;;
        tx2 = lower_bound(X, X + CX, R[i].x2) - X;;
        E[i].second = E[i + N].second = make_pair(tx1, tx2);
        E[i].first = make_pair(R[i].y1, -1);
        E[i + N].first = make_pair(R[i].y2, 1);
```

```
    }
    sort(E, E + (N << 1));
}

//求矩形并得周长，传入储存矩形的数组 R 和矩形个数 N
int perimeter(Rect *R, int N) {
    int ret = 0, i, k, prv;
    for (k = 0; k < 2; ++k) {
        discrete(R, N);
        for (prv = i = 0; i < (N << 1); ++i) {
            T.add(-E[i].first.second, E[i].second.first, E[i].second.second - 1);
            ret += abs(T.length[K] - prv);
            prv = T.length[K];
        }
        for (i = 0; i < N; ++i) {
            swap(R[i].x1, R[i].y1);
            swap(R[i].x2, R[i].y2);
        }
    }
    return ret;
}

//求矩形并的面积，传入储存矩形的数组 R 和矩形个数 N
LL area(Rect *R, int N) {
    int i, k, prv;
    LL ret = 0;
    discrete(R, N);
    prv = E[0].first.first;
    for(i = 0; i < (N << 1); ++i) {
        ret += (LL)T.length[K] * (E[i].first.first - prv);
        prv = E[i].first.first;
        T.add(-E[i].first.second, E[i].second.first, E[i].second.second - 1);
    }
    return ret;
}
int main() {}
```

## 3.12 线段树

```
//线段树
//可以处理加入边和删除边不同的情况
//incSet 和 decSeg 用于加入边
//segLen 求长度
//t 传根节点 (一律为 1)
//L0, r0 传树的节点范围 (一律为 1..t)
//L, r 传线段 (端点)
const int MAXN = 10000;
class SegTree {
    int n, cnt[MAXN], len[MAXN];

    SegTree(int t) : n(t) {
```

```
        for (int i = 1; i <= t; i++) {
            cnt[i] = len[i] = 0;
        }
    };

    void update(int t, int L, int r);
    void incSet(int t, int L0, int r0, int L, int r);
    void decSeg(int t, int L0, int r0, int L, int r);
    int segLen(int t, int L0, int r0, int L, int r);
};

inline int length(int L, int r) {
    return r - L;
}

void SegTree::update(int t, int L, int r) {
    if (cnt[t] || r - L == 1) {
        len[t] = length(L, r);
    } else {
        len[t] = len[t + t] + len[t + t + 1];
    }
}

void SegTree::incSet(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]++;
    } else {
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            incSet(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            incSet(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        if (cnt[t + t] && cnt[t + t + 1]) {
            cnt[t + t]--;
            update(t + t, L0, m0);
            cnt[t + t + 1]--;
            update(t + t + 1, m0, r0);
            cnt[t]++;
        }
    }
    update(t, L0, r0);
}

void SegTree::decSeg(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]--;
    } else if (cnt[t]) {
        cnt[t]--;
        if (L > L0) {
            incSet(t, L0, r0, L0, L);
        }
        if (r < r0) {
            incSet(t, L0, r0, r, r0);
        }
    } else {
        int m0 = (L0 + r0) >> 1;
```

```
        if (L < m0) {
            decSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            decSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
    }
    update(t, L0, r0);
}

int SegTree::segLen(int t, int L0, int r0, int L, int r) {
    if (cnt[t] || (L0 == L && r0 == r)) {
        return len[t];
    } else {
        int m0 = (L0 + r0) >> 1, ret = 0;
        if (L < m0) {
            ret += segLen(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            ret += segLen(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        return ret;
    }
}
```

## 3.13 线段树扩展

```
//线段树扩展
//可以计算长度和线段数
//可以处理加入边和删除边不同的情况
//incSeg 和 decSeg 用于加入边
//segLen 求长度, setCut 求线段数
//t 传根节点 (一律为 1)
//L0, r0 传树的节点范围 (一律为 1..t)
//L, r 传线段 (端点)
const int MAXN = 10000;
class SegTree {
public:
    int n, cnt[MAXN], len[MAXN], cut[MAXN], bl[MAXN], br[MAXN];

    SegTree(int t) : n(t) {
        for (int i = 1; i <= t; i++) {
            cnt[i] = len[i] = cut[i] = bl[i] = br[i] = 0;
        }
    };

    void update(int t, int L, int r);
    void incSeg(int t, int L0, int r0, int L, int r);
    void decSeg(int t, int L0, int r0, int L, int r);
    int segLen(int t, int L0, int r0, int L, int r);
    int setCut(int t, int L0, int r0, int L, int r);
};
```

```

inline int length(int L, int r) {
    return r - L;
}

void SegTree::update(int t, int L, int r) {
    if (cnt[t] || r - L == 1) {
        len[t] = length(L, r);
        cut[t] = bl[t] = br[t] = 1;
    } else {
        len[t] = len[t + t] + len[t + t + 1];
        cut[t] = cut[t + t] + cut[t + t + 1];
        if (br[t + t] && bl[t + t + 1]) {
            cut[t]--;
        }
        bl[t] = bl[t + t];
        br[t] = br[t + t + 1];
    }
}

void SegTree::incSeg(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]++;
    } else {
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            incSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            incSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        if (cnt[t + t] && cnt[t + t + 1]) {
            cnt[t + t]--;
            update(t + t, L0, m0);
            cnt[t + t + 1]--;
            update(t + t + 1, m0, r0);
            cnt[t]++;
        }
    }
    update(t, L0, r0);
}

void SegTree::decSeg(int t, int L0, int r0, int L, int r) {
    if (L0 == L && r0 == r) {
        cnt[t]--;
    } else if (cnt[t]) {
        cnt[t]--;
        if (L > L0) {
            incSeg(t, L0, r0, L0, L);
        }
        if (r < r0) {
            incSeg(t, L0, r0, r, r0);
        }
    } else {
        int m0 = (L0 + r0) >> 1;
        if (L < m0) {
            decSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
```

```
        if (r > m0) {
            decSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
    }
    update(t, L0, r0);
}

int SegTree::segLen(int t, int L0, int r0, int L, int r) {
    if (cnt[t] || (L0 == L && r0 == r)) {
        return len[t];
    } else {
        int m0 = (L0 + r0) >> 1, ret=0;
        if (L < m0) {
            ret += segLen(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            ret += segLen(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        return ret;
    }
}

int SegTree::setCut(int t, int L0, int r0, int L, int r) {
    if (cnt[t]) {
        return 1;
    }
    if (L0 == L && r0 == r) {
        return cut[t];
    } else {
        int m0 = (L0 + r0) >> 1, ret = 0;
        if (L < m0) {
            ret += setCut(t + t, L0, m0, L, m0 < r ? m0 : r);
        }
        if (r > m0) {
            ret += setCut(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
        }
        if (L < m0 && r > m0 && br[t + t] && bl[t + t + 1]) {
            ret--;
        }
        return ret;
    }
}
```

# Chapter 4

# 数论

## 4.1 整除规则

```
最后 n 位可以被 2^n 整除，则原数被 2^n 整除
各位数和可以被 3,9 整除，则原数被 3,9 整除
最后 n 位可以被 5^n 整除，则原数被 5^n 整除

对于其他的小素数有通用的方法：
删除最低位 (设为 d)，剩下的数减去 y*d 得到的新数被 x 整除，则原数可以被 x 整除.(此过程可以重复直到 \
数小到可以直接判)
x  y
7  2
11 1
13 9
17 5
19 17
23 16
29 26
31 3
37 11
41 4
43 30
47 14
```

## 4.2 分解质因数

```cpp
//分解质因数
//primeFactor() 传入 n, 返回不同质因数的个数
//f 存放质因数，nf 存放对应质因数的个数
//先调用 initPrime()，其中第二个 initPrime() 更快

#include <cmath>

const int PSIZE = 100000;

int plist[PSIZE], pcount = 0;

bool prime(int n) {
```

```
    if ((n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n % 5)) || (n != 7 \
&& !(n % 7))) {
        return false;
    }
    for (int i = 0; plist[i] * plist[i] <= n; ++i) {
        if (n % plist[i] == 0) {
            return false;
        }
    }
    return n > 1;
}

void initPrime() {
    plist[pcount++] = 2;
    for (int i = 3; i < 100000; i += 2) {
        if (prime(i)) {
            plist[pcount++] = i;
        }
    }
}

int primeFactor(int n, int *f, int *nf) {
    int cnt = 0;
    int n2 = (int) sqrt((double) n);
    for (int i = 0; n > 1 && plist[i] <= n2; ++i) {
        if (n % plist[i] == 0) {
            for (nf[cnt] = 0; n % plist[i] == 0; n /= plist[i]) {
                ++nf[cnt];
            }
            f[cnt++] = plist[i];
        }
    }
    if (n > 1) {
        nf[cnt] = 1;
        f[cnt++] = n;
    }
    return cnt;
}

//产生 MAXN 以内的所有素数
//note:2863311530 就是 10101010101010101010101010101010
//给所有 2 的倍数赋初值
#include <cmath>

const int MAXN = 100000000;
unsigned int plist[6000000], pcount;
unsigned int isPrime[(MAXN >> 5) + 1];

#define setbitzero(a) (isPrime[(a) >> 5] &= (~(1 << ((a) & 31))))
#define setbitone(a) (isPrime[(a) >> 5] |= (1 << ((a) & 31)))
#define ISPRIME(a) (isPrime[(a) >> 5] & (1 << ((a) & 31)))

void initPrime() {
    int i, j, m;
    int t = (MAXN >> 5) + 1;
    for (i = 0; i < t; ++i) {
```

```
        isPrime[i] = 2863311530;
    }
    plist[0] = 2;
    setbitone(2);
    setbitzero(1);
    m = (int) sqrt((double) MAXN);
    pcount = 1;
    for (i = 3; i <= m; i += 2) {
        if (ISPRIME(i)) {
            plist[pcount++] = i;
            for (j = i << 1; j <= MAXN; j += i) {
                setbitzero(j);
            }
        }
    }
    if (!(i & 1)) {
        ++i;
    }
    for (; i <= MAXN; i += 2) {
        if (ISPRIME(i)) {
            plist[pcount++] = i;
        }
    }
}

// O(n) 筛素数表
// 返回素数个数
// 素数保存在 plist 里面
// minp 存放最小的素因子
#include <cstring>

const int MAXN = 1000000;

int minp[MAXN + 1], plist[MAXN + 1];

int prime(int n = MAXN) {
    int num = 0;
    memset(minp, 0, sizeof(minp));
    for (int i = 2; i <= n; i++) {
        if (!minp[i]) plist[num++] = i, minp[i] = i;
        for (int j = 0; j < num && i * plist[j] <= n; j++) {
            minp[i * plist[j]] = plist[j];
            if (i % plist[j] == 0) break;
        }
    }
    return num;
}

// 先调用 prime 初始化
// 然后就可以调用 factorize 分解质因素
// 结果存在 p 里面，返回质因数个数
// 要保证 n >= 2
int factorize(int n, int *p) {
```

```
    int num = 0;
    while (n != 1) {
        p[num++] = minp[n];
        n /= minp[n];
    }
    return num;
}
```

## 4.3 同余方程合并

```
/*
    同余方程合并
    x % m[0] = c[0]
    x % m[1] = c[1]
    ...
    x % m[n-1] = c[n-1]
    0 <= c[i] < m[i], LCM(m[i]) < ll
    注意下标从 0 开始，返回最小非负解 x，返回值为 -1 表示无解
    可以处理 m[i] 不互素的情况，c[i] >= m[i] 将认为无解
*/
typedef long long ll;

ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b)
        return x = 1, y = 0, a;
    ll d = exgcd(b, a % b, x, y), t = x;
    y = t - (a / b) * (x = y);
    return d;
}

inline ll mod(ll x, ll y) { // y > 0
    return (x %= y) < 0 ? x + y : x;
}

ll solve(int n, ll m[], ll c[]) {
    for (int i = 0; i < n; i++)
        if (c[i] >= m[i]) return -1;
    ll ans = c[0], LCM = m[0], x, y, g;
    for (int i = 1; i < n; i++) {
        if (LCM < m[i]) // 防止溢出，如能保证 m[i] 在 int 范围内可以删去
            swap(LCM, m[i]), swap(ans, c[i]);
        g = exgcd(LCM, m[i], x, y);
        if ((c[i] - ans) % g) return -1;
        ans += LCM * mod((c[i] - ans) / g * x, m[i] / g);
        ans %= LCM *= m[i] / g;
    }
    return ans;
}
```

## 4.4 数论变换

```cpp
#include<algorithm>
using namespace std;
typedef long long LL;
LL pm(LL a,LL n,LL m) {
    LL r=1;
    for(; n; n>>=1,a=a*a%m)
        if(n&1)r=r*a%m;
    return r;
}
//和 FFT 类似，仅用于计算卷积，在 FFT 碰到精度问题或超时时可以考虑换用 NTT
//注意传入的 n 必须为 2 的幂次，并且计算的结果都是 mod P 下的结果，
//如果数域范围不够大，可以用多个 (P,g) 分别计算，再用中国剩余定理还原结果
//如果题目中模的数比较不常见，如果模的数是素数 p，且 p-1 有很多因子 2，很可能
//就要用 NTT，几个可替换的 (P,g) 对:
//(2113929217,5),(1811939329,13),(2130706433,3)
const LL P=2013265921,g=31;
void ntt(LL *a,size_t n,bool inv=false) {
    // inv 为 true 时表示逆运算;
    LL w=1,d=pm(g,(P-1)/n,P),t;
    size_t i,j,c,s;
    if(inv) {
        for(i=1,j=n-1; i<j; swap(a[i++],a[j--]));
        for(t=pm(n,P-2,P),i=0; i<n; ++i)a[i]=a[i]*t%P;
    }
    for(s=n>>1; s; s>>=w=1,d=d*d%P)
        for(c=0; c<s; ++c,w=w*d%P)
            for(i=c; i<n; i+=s<<1) {
                a[i|s]=(a[i]+P-(t=a[i|s]))*w%P;
                a[i]=(a[i]+t)%P;
            }
    for(i=1; i<n; ++i) {
        for(j=0,s=i,c=n>>1; c; c>>=1,s>>=1)j=j<<1|s&1;
        if(i<j)swap(a[i],a[j]);
    }
}
// 计算卷积姿势，
// 比如计算 a[0..n-1] 和 b[0..m-1] 的卷积:c[0..n+m-1]
// int l=2;
// for(;l<n+m-1;l<<=1);
// ntt(a,l);ntt(b,l);
// for(int i=0;i<l;++i)c[i]=a[i]*b[i]%P;
// ntt(c,l,true);
```

## 4.5 模线性方程 (组)

```
//扩展 Euclid 求解 gcd(a,b)=ax+by
int extGcd(int a, int b, int &x, int &y) {
    int t, ret;
    if (!b) {
        x = 1;
        y = 0;
        return a;
    }
    ret = extGcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - a / b * y;
    return ret;
}

//计算 m^a, O(loga), 本身没什么用, 注意这个按位处理的方法 :-P
int exponent(int m, int a) {
    int ret = 1;
    for (; a; a >>= 1, m *= m) {
        if (a & 1) {
            ret *= m;
        }
    }
    return ret;
}

//计算幂取模 a^b mod n, O(logb)
int modularExponent(int a, int b, int n) {
    //a^b mod n
    int ret = 1;
    for (; b; b >>= 1, a = (int) ((long long) a * a % n)) {
        if (b & 1) {
            ret = (int) ((long long) ret * a % n);
        }
    }
    return ret;
}

//求解模线性方程 ax=b (mod n)
//返回解的个数, 解保存在 sol[] 中
//要求 n>0, 解的范围 0..n-1
int modularLinear(int a, int b, int n, int *sol) {
    int d, e, x, y, i;
    d = extGcd(a, n, x, y);
    if (b % d) {
        return 0;
    }
    e = (x *(b / d) % n + n) % n;
    for (i = 0; i < d; i++) {
        sol[i] = (e + i *(n / d)) % n;
    }
```

```
    return d;
}

//求解模线性方程组（中国余数定理）
//  x = b[0] (mod w[0])
//  x = b[1] (mod w[1])
//  ...
//  x = b[k-1] (mod w[k-1])
//要求 w[i]>0,w[i] 与 w[j] 互质，解的范围 1..n,n=w[0]*w[1]*...*w[k-1]
int modularLinearSystem(int b[], int w[], int k) {
    int d, x, y, a = 0, m, n = 1, i;
    for (i = 0; i < k; i++) {
        n *= w[i];
    }
    for (i = 0; i < k; i++) {
        m = n / w[i];
        d = extGcd(w[i], m, x, y);
        a = (a + y * m * b[i]) % n;
    }
    return (a + n) % n;
}
```

## 4.6 欧拉函数

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b): a;
}

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

//求 1..n-1 中与 n 互质的数的个数
int eular(int n) {
    int ret = 1, i;
    for (i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            n /= i;
            ret *= i - 1;
            while (n % i == 0) {
                n /= i;
                ret *= i;
            }
        }
    }
    if (n > 1) {
        ret *= n - 1;
    }
    return ret;
}

// O(n) 求 1 到 n 的欧拉函数，顺便筛出一个素数表
```

```
// 素数保存在 plist 里面, euler 存放欧拉函数值
// 返回素数个数
#include <cstring>

const int MAXN = 1000000;

int euler[MAXN + 1], plist[MAXN + 1];

int doEuler(int n = MAXN) {
    int num = 0;
    memset(euler, 0, sizeof(euler));
    euler[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!euler[i]) plist[num++] = i, euler[i] = i - 1;
        for (int j = 0; j < num && i * plist[j] <= n; j++) {
            if (i % plist[j] == 0)
                euler[i * plist[j]] = euler[i] * plist[j];
            else
                euler[i * plist[j]] = euler[i] * (plist[j] - 1);
            if (i % plist[j] == 0) break;
        }
    }
    return num;
}
```

## 4.7 离散对数及原根

```
/*
    离散对数
    传入 a,b,p, 返回 a^x % p = b 的最小解 x, 若无解返回 -1.
    可以处理 p 不为素数的情况
    代码中把 b>=p 的情况判为无解
    算法复杂度 O( p^0.5*lg (p^0.5) )
*/
int gcd(int a,int b) {
    return b?gcd(b,a%b):a;
}

int pow_mod(int a,long long b,int p) {
    int r=1;
    if (p==1) return 0;
    for(; b; b>>=1,a=(long long)a*a%p)
        if (b&1) r=(long long)r*a%p;
    return r;
}

int phi(int n) {
    int i,m=n;
    for(i=2; n/i>=i; i++) if (n%i==0) {
            m=m/i*(i-1);
            while(n%i==0) n/=i;
        }
```

```
    if (n>1) m=m/n*(n-1);
    return m;
}

map <int,int> Mp;

int logorithm(int a,int b,int p) {
    if (b>=p) return -1;
    Mp.clear();
    int r=0,d,i,j,t1,t2,m=(int)ceil(sqrt(p)+1e-9),m1;
    if (p==1) return 0;
    for(i=0,t1=1; i<32; i++) {
        if (t1==b) return i;
        t1=(long long)t1*a%p;
    }
    for(t1=d=1; gcd((long long)t1*a%p,p)!=d; r++) {
        t1=(long long)t1*a%p;
        d=gcd(t1,p);
    }
    if (b%d!=0) return -1;
    if (t1==b) return r;
    Mp[t1]=0;
    int pre=t1;
    for(i=1; i<m; i++) {
        int tmp=1LL*pre*a%p;
        if (!Mp.count(tmp)) Mp[tmp]=i;
        pre=tmp;
        if (tmp==b) return r+i;
    }
    m1=phi(p/d);
    m1=pow_mod(a,m1-m%m1,p);
    for(i=1; i<m; i++) {
        b=(long long)b*m1%p;
        map <int,int>::iterator it=Mp.find(b);
        if (it!=Mp.end()) return i*m+it->second+r;
    }
    return -1;
}

// O(n^0.5) 进行质因数分解
// cnt 为不同质因数个数, fac 为所有不同质因数
void primeFactor(int n, int &cnt, int fac[]) {
    cnt = 0;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0)
            for (fac[cnt++] = i; n % i == 0; n /= i);
    if (n > 1)
        fac[cnt++] = n;
}

// O(n^0.5+gdlogn) 求素数最小原根 g
int primitiveRootForPrime(int n) {
    static int cnt, fac[30];
    if (n == 2) return 1;
    primeFactor(n - 1, cnt, fac);
    for (int j, i = 2; i < n; ++i) {
```

```
        for (j = 0; j < cnt && pow_mod(i, (n - 1) / fac[j], n) != 1; ++j);
        if (j >= cnt) return i;
    }
    return 0;
}

// O(n^0.5+gdlogn) 求任意正整数最小原根 g, 无原根返回 0
int primitiveRoot(int n) {
    static int cnt, fac[30];
    if (n == 2) return 1;
    if (n == 4) return 3;
    if (n % 4 == 0) return 0;
    primeFactor(n, cnt, fac);
    if (cnt > 2 || cnt == 2 && fac[0] != 2) return 0;
    int p1 = fac[0], p2 = cnt == 2 ? fac[1] : p1;
    int phi = ::phi(n);
    primeFactor(phi, cnt, fac);
    for (int j, i = 2; i < n; ++i) {
        if (i % p1 == 0 || i % p2 == 0) continue;
        for (j = 0; j < cnt && pow_mod(i, phi / fac[j], n) != 1; ++j);
        if (j >= cnt) return i;
    }
    return 0;
}
```

## 4.8 简易素数表

```
//用素数表判定素数, 先调用 initPrime
int plist[10000], pcount = 0;

bool prime(int n) {
    int i;
    if ((n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n % 5)) || (n != 7 \
&& !(n % 7))) {
        return false;
    }
    for (i = 0; plist[i] * plist[i] <= n; i++) {
        if (!(n % plist[i])) {
            return false;
        }
    }
    return n > 1;
}

void initPrime() {
    plist[pcount++] = 2;
    for (int i = 3; i < 50000; i += 2) {
        if (prime(i)) {
            plist[pcount++] = i;
        }
    }
}
```

## 4.9 阶乘最后非零位

```
//求阶乘最后非零位, 复杂度 O(nlogn)
//返回该位,n 以字符串方式传入
#include <cstring>
const int MAXN = 10000;

int lastdigit(char *buf) {
    const int mod[20] =  {
        1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2
    };
    int len = strlen(buf), a[MAXN], i, c, ret = 1;
    if (len == 1) {
        return mod[buf[0] - '0'];
    }
    for (i = 0; i < len; i++) {
        a[i] = buf[len - 1 - i] - '0';
    }
    for (; len; len -= !a[len - 1]) {
        ret = ret * mod[a[1] % 2 * 10 + a[0]] % 5;
        for (c = 0, i = len - 1; i >= 0; i--) {
            c = c * 10 + a[i];
            a[i] = c / 5;
            c %= 5;
        }
    }
    return ret + ret % 2 * 5;
}
```

## 4.10 高级素数表

```
// 质数的判定/筛选/分解 By 猛犸也钻地 @ 2012.02.21

#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;

class FastSieve {
public:
    static const int MOD = 1000000007, SIZE = 1000050;
    typedef long long LL;
    vector<int> pl,lo,eu,rv;
    // 线性筛出 [0,SIZE) 范围内的质数
    // lo[] 为最小质因子 (或其本身), eu[] 为欧拉函数值, rv[] 为关于 MOD 的逆元
    FastSieve() {
        lo=eu=rv=vector<int>(SIZE);
        lo[1]=eu[1]=rv[1]=1;
        for(int x=2; x<SIZE; x++) {
            rv[x]=rv[MOD%x]*LL(MOD-MOD/x)%MOD;
            if(!lo[x]) pl.push_back(lo[x]=x),eu[x]=x-1;
            for(size_t i=0; i<pl.size() && x*pl[i]<SIZE; i++) {
                lo[x*pl[i]]=pl[i];
```

```
                eu[x*pl[i]]=eu[x]*(pl[i]-(x%pl[i]!=0));
                if(x%pl[i]==0) break;
            }
        }
    }
    // 对 n 做质因数分解
    vector<LL> factorize(LL n) {
        vector<LL> u;
        int i,t=sqrt(n+1);
        for(i=0; pl[i]<=t; i++) if(n%pl[i]==0) {
                do {
                    n/=pl[i];
                    u.push_back(pl[i]);
                } while(n%pl[i]==0);
                t=sqrt(n+1);
            }
        if(n>1) u.push_back(n);
        return u;
    }
    // 判断 n 是否为质数
    bool prime(LL n) {
        if(n<SIZE) return n>=2 && lo[n]==n;
        int i,t=sqrt(n+1);
        for(i=0; pl[i]<=t; i++) if(n%pl[i]==0) return false;
        return true;
    }
};

class Primality {
public:
    typedef long long LL;
    // 判断 num 是否为质数
    bool miller_rabin(LL num) {
        if(num<=1) return false;
        if(num<=3) return true;
        if(~num&1) return false;
        const int u[]= {2,325,9375,28178,450775,9780504,1795265022,0};
        //const int u[]={2,3,5,7,11,13,17,19,23,29,0};
        LL e=num-1,a,c=0;
        while(~e&1) e/=2,c++;
        for(int i=0; u[i]; i++) {
            if(num<=u[i]) return true;
            a=POW(u[i],e,num);
            if(a==1) continue;
            for(int j=1; a!=num-1; j++) {
                if(j==c) return false;
                a=MUL(a,a,num);
            }
        }
        return true;
    }
    // 求一个小于 n 的因数，期望复杂度为 O(n^0.25)，当 n 为非合数时返回 n 本身
    LL pollard_rho(LL n) {
        if(n<=3 || miller_rabin(n)) return n; // 保证 n 为合数时可去掉这行
```

```
    while(1) {
        int i=1,cnt=2;
        LL x=rand()%n,y=x,c=rand()%n;
        if(!c || c==n-2) c++;
        do {
            LL u=__gcd(n-x+y,n);
            if(u>1 && u<n) return u;
            if(++i==cnt) y=x,cnt*=2;
            x=(c+MUL(x,x,n))%n;
        } while(x!=y);
    }
    return n;
}
// 使用 rho 方法对 n 做质因数分解，建议先筛去小质因数后再用此函数
vector<LL> factorize(LL n) {
    vector<LL> u;
    if(n>1) u.push_back(n);
    for(size_t i=0; i<u.size(); i++) {
        LL x=pollard_rho(u[i]);
        if(x==u[i]) continue;
        u[i--]/=x;
        u.push_back(x);
    }
    sort(u.begin(),u.end());
    return u;
}
// 返回 x 与 y 相乘模 m 的结果，x 与 m 要小于 2^62
LL MUL(LL a, LL b, LL mod) { //O(1) long long 乘法
    assert(0 <= a && a < mod);
    assert(0 <= b && b < mod);
    if (mod < int(1e9)) return a * b % mod;
    LL k = (LL)((long double)a * b / mod);
    LL res = a * b - k * mod;
    res %= mod;
    if (res < 0) res += mod;
    return res;
}
/* 普通的快速乘
LL MUL(LL x, LL y, LL m){
    LL c,s=0;
    for(c=x%m;y;c=(c+c)%m,y>>=1)
        if(y&1) s=(s+c)%m;
    return s;
}*/
// 返回 x 的 y 次方模 m 的结果，x 与 m 要小于 2^62
LL POW(LL x, LL y, LL m) {
    LL c,s=1;
    for(c=x%m; y; c=MUL(c,c,m),y>>=1)
        if(y&1) s=MUL(s,c,m);
    return s;
}
```

```
};
```

# Chapter 5

# 数值

## 5.1 FFT

```
//用傅立叶变换可在 O(nlogn) 求卷积
//传入长度为 len(保证为 2 的幂) 的数组 a,inv 为 1 和 -1 时分别做正反 DFT 变换
//常数较大
#include <complex>
void FFT(complex<double> *a,int len,int inv) { //eps=1e-12
    for (int i=0, n1=0, n2=0; i < len; ++i, n2 ^= (len/(i&-i)>>1), n1^=(i&-i))
        if (n1 > n2)
            swap(a[n1], a[n2]);
    for(int m = 1; m <= len >> 1; m <<= 1) {
        complex<double> w0(cos(PI / m), sin(PI / (inv * m))), w = 1, t;
        for(int k = 0; k < len; k += (m << 1), w = 1)
            for(int j = 0; j < m; ++j, w *= w0) {
                t=w*a[k+j+m];
                a[k+j+m]=a[k+j]-t;
                a[k+j]+=t;
            }
    }
    if(inv == -1)
        for(int i = 0; i < len; ++i)
            a[i] /= len;
}
```

## 5.2 周期性方程 (追赶法)

```
/*  追赶法解周期性方程，复杂度 o(n)
    周期性方程定义: | b0 c0 ...                 a0   |          =  x0
                    | a1 b1 c1 ...                  |          =  x1
                    |       ...                     | *   X    =  ...
                    |     ...         an-2 bn-2 cn-2 |          =  xn-2
                    | cn-1                an-1 bn-1 |          =  xn-1
    输入: a[],b[],c[],x[],n
    输出: 求解结果 X 在 x[] 中
```

```
*/

void run(double a[], double b[], double c[], double x[], int n) {
    c[0] /= b[0];
    a[0] /= b[0];
    x[0] /= b[0];
    for (int i = 1; i < n - 1; i++) {
        double temp = b[i] - a[i] * c[i - 1];
        c[i] /= temp;
        x[i] = (x[i] - a[i] * x[i - 1]) / temp;
        a[i] = -a[i] * a[i - 1] / temp;
    }
    a[n - 2] = -a[n - 2] - c[n - 2];
    for (int i = n - 3; i >= 0; i--) {
        a[i] = -a[i] - c[i] * a[i + 1];
        x[i] -= c[i] * x[i + 1];
    }
    x[n - 1] -= (c[n - 1] * x[0] + a[n - 1] * x[n - 2]);
    x[n - 1] /= (c[n - 1] * a[0] + a[n - 1] * a[n - 2] + b[n - 1]);
    for (int i = n - 2; i >= 0; i--) {
        x[i] += a[i] * x[n - 1];
    }
}
```

## 5.3 多项式求根 (牛顿法)

```
/* 牛顿法解多项式的根
   输入：多项式系数 c[]，多项式度数 n，求在 [a,b] 间的根
   输出：根
   要求保证 [a,b] 间有根
*/
#include <cmath>
#include <cstdlib>

double f(int m, double c[], double x) {
    int i;
    double p = c[m];
    for (i = m; i > 0; i--) {
        p = p * x + c[i - 1];
    }
    return p;
}

int newton(double x0, double *r, double c[], double cp[], int n, double a, double b, dou\
ble eps) {
    const int MAX_ITERATION = 1000;
    int i = 1;
    double x1, x2, fp, eps2 = eps / 10.0;
    x1 = x0;
    while (i < MAX_ITERATION) {
        x2 = f(n, c, x1);
        fp = f(n - 1, cp, x1);
        if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0)) {
```

```
            return 0;
        }
        x2 = x1 - x2 / fp;
        if (fabs(x1 - x2) < eps2) {
            if (x2 < a || x2 > b) {
                return 0;
            }
            *r = x2;
            return 1;
        }
        x1 = x2;
        i++;
    }
    return 0;
}

double polynomialRoot(double c[], int n, double a, double b, double eps) {
    double *cp;
    int i;
    double root;

    cp = (double *) calloc(n, sizeof(double));
    for (i = n - 1; i >= 0; i--) {
        cp[i] = (i + 1) * c[i + 1];
    }
    if (a > b) {
        root = a;
        a = b;
        b = root;
    }
    if ((!newton(a, &root, c, cp, n, a, b, eps)) && (!newton(b, &root, c, cp, n, a, b, e\
ps))) {
        newton((a + b) * 0.5, &root, c, cp, n, a, b, eps);
    }
    free(cp);
    if (fabs(root) < eps) {
        return fabs(root);
    } else {
        return root;
    }
}
```

## 5.4 定积分计算 (Romberg)

```
/*  Romberg 求定积分
    输入：积分区间 [a,b]，被积函数 f(x,y,z)
    输出：积分结果

    f(x,y,z) 示例：
    double f0( double x, double l, double t )
    {
        return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
```

```
    }
*/
#include <cmath>

double romberg(double a, double b, double(*f)(double x, double y, double z), double eps,\
 double l, double t) {
    const int MAXN = 1000;
    int i, j, temp2, min;
    double h, R[2][MAXN], temp4;

    for (i = 0; i < MAXN; i++) {
        R[0][i] = 0.0;
        R[1][i] = 0.0;
    }
    h = b - a;
    min = (int)(log(h * 10.0) / log(2.0)); //h should be at most 0.1
    R[0][0] = ((*f)(a, l, t) + (*f)(b, l, t)) * h * 0.50;
    i = 1;
    temp2 = 1;
    while (i < MAXN) {
        i++;
        R[1][0] = 0.0;
        for (j = 1; j <= temp2; j++) {
            R[1][0] += (*f)(a + h *((double)j - 0.50), l, t);
        }
        R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
        temp4 = 4.0;
        for (j = 1; j < i; j++) {
            R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j - 1]) / (temp4 - 1.0);
            temp4 *= 4.0;
        }
        if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i > min)) {
            return R[1][i - 1];
        }
        h *= 0.50;
        temp2 *= 2;
        for (j = 0; j < i; j++) {
            R[0][j] = R[1][j];
        }
    }
    return R[1][MAXN - 1];
}

double integral(double a, double b, double(*f)(double x, double y, double z), double eps\
, double l, double t) {
    const double PI = 3.1415926535897932;
    int n;
    double R, p, res;

    n = (int)(floor)(b * t * 0.50 / PI);
    p = 2.0 * PI / t;
    res = b - (double)n * p;
    if (n) {
        R = romberg(a, p, f, eps / (double)n, l, t);
    }
    R = R *(double)n + romberg(0.0, res, f, eps, l, t);
```

```

    return R / 100.0;
}

// 其实不妨先考虑用复化 Simpson 公式
// S = h / 6 * [f(A) + 4 * ∑f(Xk+1/2) + 2 * ∑f(Xk) + f(B)]; k = 0..n-1
```

## 5.5 定积分计算 (变步长 simpson)

```
//变步长辛普森积分
double func(double x) {
    return ...; //根据题目需要实现
}
double Simpson_VariStep(double x1,double xh,double eps) {
    int subs=1,n=1,i;
    double result,x,p,width=xh-x1,t1=width*(func(x1)+func(xh))/2.0,t2;
    double s1=t1,s2=s1+2.0*eps;
    while(subs) {
        for (p=0.0,i=0; i<=n-1; ++i) {
            x=x1+(i+0.5)*width;
            p=p+func(x);
        }
        t2=(t1+width*p)/2.0;
        s2=(t2*4-t1)/3.0;
        result=s2;
        subs=(fabs(s2-s1)>=eps);
        t1=t2;
        s1=s2;
        n+=n;
        width=width/2.0;
    }
    return s2;
}
```

## 5.6 定积分计算 (自适应 simpson)

```
#include<cmath>
using namespace std;
typedef double flt;
//用于无法确定如何分割区间使得积分精确的情况。
flt f(flt x) {
    return x; //被积函数，根据需要实现
}
flt adaptive(flt l,flt r,flt m,flt eps,flt s,flt fl,flt fr,flt fm,int lim) {
    flt h=(r-l)/12;
    flt u=ldexp(l+m,-1),v=ldexp(m+r,-1);
    flt fu=f(u),fv=f(v);
    flt sl=(fl+fm+ldexp(fu,2))*h,
        sr=(fm+fr+ldexp(fv,2))*h,
        s2=sl+sr;
    if(lim<=0 || fabs(s2-s)<= eps)
```

```cpp
        return s2+(s2-s)/15;
    eps=ldexp(eps,-1);
    --lim;
    return  adaptive(l,m,u,eps,sl,fl,fm,fu,lim)
            +   adaptive(m,r,v,eps,sr,fm,fr,fv,lim);
}
// eps 是需要精度, maxrec 是限制最大迭代次数.
// 积分区间 [l,r]
flt simpson(flt l,flt r,flt eps=1e-6,int maxrec=16) {
    flt m=ldexp(l+r,-1);
    flt fl=f(l),fm=f(m),fr=f(r);
    return adaptive(l,r,m,eps*15,(r-l)/6*(fl+fr+ldexp(fm,2)),fl,fr,fm,maxrec);
}
```

## 5.7 线性相关

```cpp
//判线性相关(正交化)
//传入 m 个 n 维向量
#include <cmath>
const int MAXN = 100;
const double EPS = 1e-10;

bool linearDependent(int m, int n, double vec[][MAXN]) {
    double ort[MAXN][MAXN], e;
    int i, j, k;
    if (m > n) {
        return true;
    }
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            ort[i][j] = vec[i][j];
        }
        for (k = 0; k < i; k++) {
            for (e = j = 0; j < n; j++) {
                e += ort[i][j] * ort[k][j];
            }
            for (j = 0; j < n; j++) {
                ort[i][j] -= e * ort[k][j];
            }
            for (e = j = 0; j < n; j++) {
                e += ort[i][j] * ort[i][j];
            }
            if (fabs(e = sqrt(e)) < EPS) {
                return 1;
            }
            for (j = 0; j < n; j++) {
                ort[i][j] /= e;
            }
        }
    }
    return false;
}
```

## 5.8 线性规划

```
#include<algorithm>
#include<cmath>
#include<cstring>
using namespace std;
typedef double dbl;
const dbl inf = 1e99,eps = 1e-8;
//注意此 LP 处理的情况都是 xi 是非负的结果
//如果 xi 是无限制, 则用 xi = xj-xk 去变换
//如果 xi 是小于某个数 V, 则用 xi=V-xj 去变换
struct LP {
    static const size_t N=20,M=20,L=N+M*2,faux=~0;
    dbl A[M][L],b[M],c[N+M],c1[L],x[N],d[L],zm;
    size_t bs[M],n,m,l;
#define go(i,n) for(i=0;i<n;++i)
    dbl simplex(dbl *c) {
        for(size_t j,k,i,o;; bs[o]=i) {
            dbl dmax=0,u=inf,t;
            i=o=-1;
            go(k,l) {
                d[k]=c[k];
                go(j,m)d[k]-=c[bs[j]]*A[j][k];
                if(d[k]>dmax+eps) {
                    dmax=d[k];
                    i=k;
                }
            }
            if(!~i) {
                zm=0;
                fill(x,x+n,0);
                go(j,m)if(bs[j]<n)
                    zm+=c[bs[j]]*(x[bs[j]]=b[j]);
                return zm;
            }
            go(j,m)if(A[j][i]>eps&&b[j]/A[j][i]+eps<u)
                u=b[o=j]/A[j][i];
            if(!~o)throw "unbounded!"; //可行域无界, 无最大值
            b[o]*=t=1/A[o][i];
            go(k,l)A[o][k]*=t;
            go(j,m)if(j!=o&&fabs(A[j][i])>eps) {
                t=-A[j][i];
                b[j]+=t*b[o];
                go(k,l)A[j][k]+=t*A[o][k];
            }
        }
    }
    void init(int _n) {
        memset(A,0,sizeof(A)); //方程左边系数
        memset(b,0,sizeof(b)); //方程右边的值
        memset(c,0,sizeof(c)); //目标函数系数
        l=n=_n;
        m=0;
```

```
    }
    void add_contraint(dbl *a,dbl _b,bool eq=false) {
        // eq 为 false: 添加约束为不等式 a0*x0+a1*x1+...+ <= _b
        // eq 为 true: 添加约束为等式 a0*x0+a1*x1+ ... + == _b
        copy(a,a+n,A[m]);
        b[m]=_b;
        if(eq||_b<0)bs[m]=faux;
        else A[m][bs[m]=l++]=1;
        m++;
    }
    dbl solve() {
        size_t ol=l,j,k;
        go(j,m)if(bs[j]==faux)A[j][bs[j]=l++]=1;
        if(ol!=l) {
            go(k,l)c1[k]=k<ol?0:-1;
            if(simplex(c1)<-eps)throw "infeasible"; //无可行解
            l=ol;
        }
        return simplex(c);
    }
};
```

## 5.9 高斯消元 (全主元)

```
#include <cmath>

const int MAXN = 100;
const double EPS = 1e-10;

//全主元 gauss 消去解 a[][]x[]=b[]
//返回是否有唯一解, 若有解在 b[] 中
bool gaussTpivot(int n, double a[][MAXN], double b[]) {
    int i, j, k, row, col, index[MAXN];
    double maxp, t;
    for (i = 0; i < n; i++) {
        index[i] = i;
    }
    for (k = 0; k < n; k++) {
        for (maxp = 0, i = k; i < n; i++) {
            for (j = k; j < n; j++) {
                if (fabs(a[i][j]) > fabs(maxp)) {
                    maxp = a[row = i][col = j];
                }
            }
        }
        if (fabs(maxp) < EPS) {
            return false;
        }
        if (col != k) {
            for (i = 0; i < n; i++) {
                swap(a[i][col], a[i][k]);
            }
            swap(index[col],  index[k]);
```

```
        }
        if (row != k) {
            for (j = k; j < n; j++) {
                swap(a[k][j], a[row][j]);
            }
            swap(b[k], b[row]);
        }
        for (j = k + 1; j < n; j++) {
            a[k][j] /= maxp;
            for (i = k + 1; i < n; i++) {
                a[i][j] -= a[i][k] * a[k][j];
            }
        }
        b[k] /= maxp;
        for (i = k + 1; i < n; i++) {
            b[i] -= b[k] * a[i][k];
        }
    }
    for (i = n - 1; i >= 0; i--) {
        for (j = i + 1; j < n; j++) {
            b[i] -= a[i][j] * b[j];
        }
    }
    for (k = 0; k < n; k++) {
        a[0][index[k]] = b[k];
    }
    for (k = 0; k < n; k++) {
        b[k] = a[0][k];
    }
    return true;
}
```

## 5.10 高斯消元 (列主元)

```
//列主元法高斯消元, 矩阵存在 e 数组中
//n 行 m+1 列的矩阵, 第 m+1 列为等式右侧的常量
//返回值 -1 表示无解, 其他数字表示矩阵的秩
//有解时返回任意一组解, 在 x[0..m-1] 中
//pos[i] 表示第 i 行第一个非零数的位置
//复杂度 O(min(n,m)*n*m)
typedef double flt;
const flt eps=1e-9; //有的范围大的题目可能需要更小的 eps
const int MAXN=512, MAXM=512;
struct Gauss {
    flt e[MAXN][MAXM],x[MAXM];
    int pos[MAXM];
    int solve(int n, int m) {
        int r=0;
        //c for column, r for row
        for(int c=0,i,j,k; c<=m&&r<n; ++c) {
            for(i=k=r; i<n; ++i)
                if(fabs(e[i][c])>fabs(e[k][c]))k=i;
```

```
            if(k!=r)for(j=c; j<=m; ++j)swap(e[k][j],e[r][j]);
            if(fabs(e[r][c])<eps)continue;
            pos[r]=c;
            for(i=r+1; i<n; ++i) {
                flt t=-e[i][c]/e[r][c];
                for(int j=c; j<=m; ++j)e[i][j]+=t*e[r][j];
            }
            ++r; //注意, 前面有 continue, 不能写到 for 里面去
        }
        if(r>0 && pos[r-1]==m)return -1;
        for(int i=0; i<m; ++i)x[i]=0;
        for(int i=r-1; i>=0; --i) {
            int c=pos[i];
            x[c]=e[i][m]/e[i][c];
            for(int j=0; j<i; ++j)e[j][m]-=e[j][c]*x[c];
        }
        return r;
    }
};
```

# Chapter 6

# 字符串

## 6.1 Trie 图 (dd engi)

```cpp
/*
    Trie 图，即用于多串匹配的字符串自动机。
    对于字符串 S 中是否存在模式串 s_1,s_2,..,s_n 的匹配的问题，
    可以用 O((|s_1|+|s_2|+...+|s_n|)|Σ|) 的时间预处理，O(|S|) 的时间回答询问。
    若深入理解，也可根据具体情况扩展之，以解决很多字符串有关的题。
    注意事项：
        1. 字符的类型、字符集的大小可改。
        2. ELEMENT_MAX 一般可设为 |s_1|+|s_2|+...+|s_n| 的最大可能值。
        3. 传递的字符串中每个字符应在 [0,SIGMA) 的区间内。
    使用方法：
        1. 处理每个测试点前调用 init() 初始化。
        2. 用 insert 插入 s_1,s_2,...
        3. 调用 build_graph 建立 trie 图。
        4. 调用 match 查询 S 中是否存在匹配。
*/
#include <cstring>
#include <queue>
#include <cstdio>
using namespace std;
typedef struct node *trie;
const int SIGMA = 26;
const int ELEMENT_MAX = 50000;
int tot;
struct node {
    bool match;
    trie pre, child[SIGMA];
} T[ELEMENT_MAX];

void trie_init() {
    tot = 1;
    memset(T, 0, sizeof(T));
```

```
}
void insert(char *s, int n) {
    trie t = T;
    for (int i = 0; i < n; ++i) {
        int c = s[i] - 'a';
        if (!t->child[c]) {
            t->child[c] = &T[tot++];
        }
        t = t->child[c];
    }
    t->match = true;
}
void build_graph() {
    trie t = T;
    queue <trie> Q;
    for (int i = 0; i < SIGMA; ++i) {
        if (t->child[i]) {
            t->child[i]->pre = t;
            Q.push(t->child[i]);
        } else {
            t->child[i] = t;
        }
    }
    while (!Q.empty()) {
        t = Q.front();
        Q.pop();
        t->match |= t->pre->match;
        for (int i = 0; i < SIGMA; ++i) {
            if (t->child[i]) {
                t->child[i]->pre = t->pre->child[i];
                Q.push(t->child[i]);
            } else {
                t->child[i] = t->pre->child[i];
            }
        }
    }
}
bool match(char *s, int n) {
    trie t = T;
    for (int i = 0; i < n; ++i) {
        int c = s[i] - 'a';
        t = t->child[c];
    }
    return t->match;
}
//示例程序
int main() {
    trie_init();
    insert("abcd", 4);
    insert("bc", 2);
    build_graph();

    printf("%s\n", match("abc", 3) ? "Yes" : "No"); //output: Yes
}
```

## 6.2 Trie 图 (猛犸也钻地)

```
// 确定性 AC 自动机 (Trie 图) By 猛犸也钻地 @ 2011.11.24

#include <cstring>
#include <algorithm>
using namespace std;

class TrieGraph {
public:
    static const int SIZE = 100005; // 最大结点总数，约为模板串长度之和
    static const int LEAF = 26;     // 每个结点下的叶子数量
    // next[] 指向了含有相同后缀但更短的一个字符串，n 为当前存在的结点总数
    int next[SIZE],e[SIZE][LEAF],n; // e[][] 为结点的各个叶子的编号
    int data[SIZE]; // data[] 一般用位标记维护当前的串匹配上了哪些模式串
    TrieGraph() {
        n=SIZE;    // 别忘了写上这行
    }
    void init() {
        fill_n(next,n,0);
        fill_n(data,n,0);
        memset(e,-1,n*sizeof(e[0]));
        n=1;
    }
    void insert(const char *s, int idx = 0) {
        int x=0;
        for(int i=0; s[i]; i++) {
            int c=s[i]-'a'; // 根据题目的字符集修改这里的映射方式
            x=~e[x][c]?e[x][c]:e[x][c]=n++;
        }
        data[x]|=1<<idx;
    }
    void make() {
        static int q[SIZE],m;
        next[0]=m=0;
        for(int c=0; c<LEAF; c++)
            if(~e[0][c]) next[q[m++]=e[0][c]]=0;
            else e[0][c]=0;
        for(int i=0; i<m; i++) {
            int x=q[i];
            data[x]|=data[next[x]]; // 求 next[] 路径上的前缀和
            for(int c=0; c<LEAF; c++) {
                int t=e[next[x]][c];
                if(~e[x][c]) next[q[m++]=e[x][c]]=t;
                else e[x][c]=t;
            }
        }
    }
};
```

## 6.3 后缀数组 -线性

```
namespace SA {
int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
#define pushS(x) sa[cur[s[x]]--] = x
#define pushL(x) sa[cur[s[x]]++] = x
#define inducedSort(v) fill_n(sa, n, -1); fill_n(cnt, m, 0);                  \
    for (int i = 0; i < n; i++) cnt[s[i]]++;                                  \
    for (int i = 1; i < m; i++) cnt[i] += cnt[i-1];                           \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
    for (int i = n1-1; ~i; i--) pushS(v[i]);                                  \
    for (int i = 1; i < m; i++) cur[i] = cnt[i-1];                            \
    for (int i = 0; i < n; i++) if (sa[i] > 0 &&  t[sa[i]-1]) pushL(sa[i]-1); \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
    for (int i = n-1;  ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
void sais(int n, int m, int *s, int *t, int *p) {
    int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
    for (int i = n-2; ~i; i--) t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
    for (int i = 1; i < n; i++) rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
    inducedSort(p);
    for (int i = 0, x, y; i < n; i++) if (~(x = rk[sa[i]])) {
            if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ch++;
            else for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++)
                    if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {
                        ch++;
                        break;
                    }
            s1[y = x] = ch;
        }
    if (ch+1 < n1) sais(n1, ch+1, s1, t+n, p+n1);
    else for (int i = 0; i < n1; i++) sa[s1[i]] = i;
    for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
    inducedSort(s1);
}
template<typename T>
int mapCharToInt(int n, const T *str) {
    int m = *max_element(str, str+n);
    fill_n(rk, m+1, 0);
    for (int i = 0; i < n; i++) rk[str[i]] = 1;
    for (int i = 0; i < m; i++) rk[i+1] += rk[i];
    for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
    return rk[m];
}
// Ensure that str[n] is the unique lexicographically smallest character in str.
template<typename T>
void suffixArray(int n, const T *str) {
    int m = mapCharToInt(++n, str);
    sais(n, m, s, t, p);
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
        int j = sa[rk[i]-1];
        while (i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
        if (ht[rk[i]] = h) h--;
    }
}
};
```

## 6.4 后缀数组

```cpp
#include <bits/stdc++.h>
using namespace std;
// partial_sum 在头文件 numeric 中
// 数组下标后后缀都从 0 开始标号, 空串不作考虑
// sa[i] 表示第 i 小的后缀从字符串的第 sa[i] 个位置开始
// rk[i] 表示字符串第 i 个位置开始的后缀是第 rk[i] 小的字符串
// ht[i] 表示 sa[i] 和 sa[i-1] 所代表字符串的公共前缀的长度
struct SuffixArray {
public:
    const static int MAXN = 100000 + 10;
    int cnt[MAXN], tr[2][MAXN], ts[MAXN];
    int sa[MAXN], rk[MAXN], ht[MAXN], len; //len 字符串长度
    // 对字符串 s[0..n-1] 做后缀排序, s[n] 最好为'\0'
    // 排序完, sa[0..n-1] 为有效后缀, 空串不作考虑
    void construct(const char *s, int n, int m=256) {
        int i,j,k,*x=tr[0],*y=tr[1];
        this->len = n;
        memset(cnt,0,sizeof(cnt[0])*m);
        for (i=0; i<n; ++i) cnt[s[i]]++;
        partial_sum(cnt,cnt+m,cnt);
        for (i=0; i<n; ++i) rk[i]=cnt[s[i]]-1;
        for (k=1; k<=n; k<<=1) {
            for (i=0; i<n; ++i) x[i]=rk[i],y[i]=i+k<n?rk[i+k]+1:0;
            fill(cnt,cnt+n+1,0);
            for (i=0; i<n; ++i) cnt[y[i]]++;
            partial_sum(cnt,cnt+n+1,cnt);
            for (i=n-1; i>=0; --i) ts[--cnt[y[i]]]=i;
            fill(cnt,cnt+n+1,0);
            for (i=0; i<n; ++i) cnt[x[i]]++;
            partial_sum(cnt,cnt+n+1,cnt);
            for (i=n-1; i>=0; --i) sa[--cnt[x[ts[i]]]]=ts[i];
            for (i=rk[sa[0]]=0; i+1<n; ++i) {
                rk[sa[i+1]]=rk[sa[i]]+(x[sa[i]]!=x[sa[i+1]]||y[sa[i]]!=y[sa[i+1]]);
            }
        }
        for (i=0,k=0; i<n; ++i) {
            if (!rk[i]) continue;
            for (j=sa[rk[i]-1]; i+k<n&&j+k<n&&s[i+k]==s[j+k];) k++;
            ht[rk[i]]=k;
            if (k) k--;
        }
        rmq_init(n);
    }
    // 求任意两个后缀的 lcp
    inline int lcp(int a, int b) {
        a=rk[a],b=rk[b];
        if (a == b) return len - a + 1;
        if (a>b) swap(a,b);
        return rmq(a+1,b);
    }
```

```
private:
    int mx[MAXN][20], LOG[MAXN];
    void rmq_init(int n) {
        for (int i=-(LOG[0]=-1); i<n; ++i) LOG[i]=LOG[i>>1]+1;
        for (int i=0; i<n; ++i) mx[i][0]=ht[i];
        for (int i,j=1; (1<<j)<n; ++j) {
            for (i=0; i+(1<<j)<=n; ++i)
                mx[i][j]=min(mx[i][j-1],mx[i+(1<<(j-1))][j-1]);
        }
    }
    inline int rmq(int a, int b) {
        int k=LOG[b-a+1];
        return min(mx[a][k],mx[b-(1<<k)+1][k]);
    }
};
```

## 6.5 后缀自动机

```
/*
这是求解多串 LCS 的例程
使用时先调用 init()，再依次在线调用 add 就可以构建出对应串的后缀自动机（注意这是个在线过程）
每次 add 至多加入两个点
大致来说建成的后缀自动机有这么几个性质
1、对于任意一个给定串的后缀，在上面按顺序转移，最终一定会转移到 last。所以任意一个子串在 \
上面跑，不可能遇到 null 边
2、既然满足 1，那么显然跑到某一个状态以后，后续可接收的串的集合必然一致。
3、val 表示的是能从 root 转移到这个状态的最长串的长度
4、而能 root 转移到本状态的串的长度实际是在这个区间内 [this->fa->val + 1, this->val]
5、this 能接收的字符串的集合是 fa 对应的结点能接收的字符串的集合的子集。
6、不会存在状态 p，this 可接收的字符串集合是 p 可以接收的字符串集合的子集，而且 p 对应集合的势 \
比 fa 的要小
*/

template <class T> void checkmin(T &t,T x) {
    if (x < t) t = x;
}
template <class T> void checkmax(T &t,T x) {
    if (x > t) t = x;
}
#define foreach(it,v) for (__typeof((v).begin()) it = (v).begin();it != (v).end();it++)
const int N = 250005;

struct Node {
    Node *ch[26], *fa;
    int val;
    int len[10];
    Node():
        val(0), fa(NULL) {
        memset(ch, 0, sizeof(ch));
```

```
        memset(len, 0, sizeof(len));
    }
} pool[N * 2 + 5], *last, *root;
vector <Node *> vec[N];

namespace SAM {
int cnt;

void init() {
    if (cnt)
        for (int i = 0; i < cnt; i++)
            pool[i] = Node();
    cnt = 1;
    root = &pool[0];
    last = root;
}

void add(int c) {
    Node *p = last, *np = &pool[cnt++];
    last = np;
    np->val = p->val + 1;
    for  (; p && !p->ch[c]; p = p->fa)
        p->ch[c] = np;
    if (!p) {
        np->fa = root;
    } else {
        Node *q = p->ch[c];
        if (p->val + 1 == q->val) {
            np->fa = q;
        } else {
            Node *nq = &pool[cnt++];
            *nq = *q;
            nq->val = p->val + 1;
            q->fa = nq;
            np->fa = nq;
            for (; p && p->ch[c] == q; p = p->fa)
                p->ch[c] = nq;
        }
    }
}
}

int m, n;
char S[N], T[N];

int main() {
    SAM::init();
    scanf("%s", S);
    m = strlen(S);
    for (int i = 0; i < m; i++)
        SAM::add(S[i] - 'a');
    int k;
    for (k = 0; scanf("%s", T) != EOF; k++) {
        Node *p = root;
        int cnt = 0;
        n = strlen(T);
        for (int i = 0; i < n; i++) {
            int c = T[i] - 'a';
```

```
            if (p->ch[c]) {
                cnt++;
                p = p->ch[c];
            } else {
                for (; p && !p->ch[c]; p = p->fa);
                if (!p) {
                    p = root;
                    cnt = 0;
                } else {
                    cnt = p->val + 1;
                    p = p->ch[c];
                }
            }
            checkmax(p->len[k], cnt);
        }
    }
    for (int i = 0; i < SAM::cnt; i++) {
        vec[pool[i].val].push_back(&pool[i]);
    }
    int ans = 0;
    for (int i = m; i >= 0; i--) {
        foreach (it, vec[i]) {
            Node *p = *it;
            int now = p->val;
            for (int j = 0; j < k; j++) {
                checkmin(now, p->len[j]);
                if (p->fa) {
                    checkmax(p->fa->len[j], p->len[j]);
                }
            }
            checkmax(ans, now);
        }
    }
    printf("%d\n", ans);
}
```

## 6.6 回文树

```
#include <bits/stdc++.h>
using namespace std;
// palindromic tree, O(N)
// 每个节点表示一个回文串, len 表示回文串长度, cnt 表示回文串出现次数, fa 表示最长回文后缀
// 这棵树有两个根 node 0 和 node 1, node 0 是奇数长度回文串的根, node 1 是偶数长度回文串的根
// 首先调用 PT::init() 传入字符串 p 和字符串长度 n
// 这是一个在线过程, 每次调用 PT::add(i) 添加字符, 返回 true 表示新增一个本质不同的回文串
// 添加完成之后, 调用 PT::count(), 计算每个回文串出现次数
// 功能:
// 1. 统计字符串每个前缀中本质不同的回文串个数, 每次 add() 之后, PT::sz-2 就是答案
// 2. 统计以 i 结尾的回文串个数, 每次 add() 之后, PT::last->num 就是答案
// 3. 统计出现次数最多的回文串, max(pool[i].cnt)
```

```cpp
    // 4. 遍历所有回文串奇数长度从 node 0 开始, 偶数长度从 node 1 开始
namespace PT {
static const int MAXN = 300000 + 10, SIGMA = 26;
struct Node {
    Node *ch[SIGMA], *fa;
    int len, num, cnt;
} pool[MAXN], *last; //last 当前串的最长回文后缀
int sz;
char *s;
void init(char p[], int n) {
    memset(pool, 0, sizeof(pool[0]) * (n + 10));
    s = p;
    last = &pool[1];
    sz = 2;
    pool[0].len = -1;
    pool[0].fa = &pool[0];
    pool[1].len = 0;
    pool[1].fa = &pool[0];
}
bool add(int pos) {
    Node *cur = last;
    int curlen = 0, c = s[pos] - 'a';
    while (1) {
        curlen = cur->len;
        if (pos >= 1 + curlen && s[pos - 1 - curlen] == s[pos]) break;
        cur = cur->fa;
    }
    if (cur->ch[c]) return last = cur->ch[c], last->cnt ++, false;
    last = &pool[sz ++];
    last->cnt ++;
    last->len = cur->len + 2;
    cur->ch[c] = last;
    if (last->len == 1) return last->fa = &pool[1], last->num = 1;
    while (1) {
        cur = cur->fa;
        curlen = cur->len;
        if (pos >= curlen + 1 && s[pos - 1 - curlen] == s[pos]) {
            last->fa = cur->ch[c];
            break;
        }
    }
    return last->num = last->fa->num + 1;
}
void count() {
    for (int i = sz - 1; i > 1; -- i) pool[i].fa->cnt += pool[i].cnt;
}
}

const int MAXN = 300000 + 10;
char s[MAXN];

int main() {
    scanf("%s", s);
    int len = strlen(s);
    PT::init(s, len);
    // total distinct palindromic string of s is <= len
```

```
    // new add a letter will add 0 or 1 new palindromic string
    for (int i = 0; i < len; ++ i) {
        PT::add(i); // true: add a new palindromic string
        // cout << PT::last->num; // number of palindrome ends with s[i]
        // cout << PT::sz - 2 << endl; // number of distinct palindrome in prefix [0..i-\
1]
        // cout << ret << endl; // total palindrome in prefix [0..i-1]
    }
    PT::count(); // calc occurrence of each palindrome
    // max(PT::pool[i].cnt * PT::pool[i].len)
    return 0;
}
```

## 6.7 字符串最小表示

```
/*
    求字符串的最小表示
    输入：字符串
    返回：字符串最小表示的首字母位置 (0...size-1)
*/
#include <vector>
using namespace std;

template <class T>
int minString(const vector<T>   &str) {
    int i, j, k;
    vector<T> ss(str.size() << 1);
    for (i = 0; i<str.size(); i++) {
        ss[i] = ss[i + str.size()] = str[i];
    }
    for (i = k = 0, j = 1; k<str.size() && i<str.size() && j<str.size();) {
        for (k = 0; k<str.size() && ss[i + k] == ss[j + k]; k++);
        if (k<str.size()) {
            if (ss[i + k]> ss[j + k]) {
                i += k + 1;
            } else {
                j += k + 1;
            }
            if (i == j) {
                j++;
            }
        }
    }
    return i < j ? i : j;
}
```

## 6.8 最长回文子串

```cpp
// 最长回文子串 (Manacher) By 猛犸也钻地 @ 2012.11.29

#include <vector>
#include <algorithm>
using namespace std;

// 传入字符串 s 和长度 n，返回最长回文子串的直径，复杂度 O(n)
int manacher(const char *s, int n) {
    vector<int> u(n+n-1,1); // u[i] 表示以 i/2 为圆心的最长回文子串的直径
    for(int i=1,x=0; i<n+n-1; i++) { // 比如字符串 babbaa，看作 b.a.b.b.a.a
        u[i]=max(x+u[x]-i,1-i%2); // 相应位置的直径长度就是 10301410121
        if(x+x>=i) u[i]=min(u[i],u[x+x-i]);
        int a=(i-1-u[i])>>1,b=(i+1+u[i])>>1;
        while(a>=0 && b<n && s[a]==s[b]) a--,b++,u[i]+=2;
        if(i+u[i]>x+u[x]) x=i;
    }
    return *max_element(u.begin(),u.end());
}
```

## 6.9 模式匹配 (KMP+Z)

```cpp
// 字符串前缀匹配 (KMP) 和后缀匹配 (Z-Function) By 猛犸也钻地 @ 2012.02.02

#include <vector>
#include <algorithm>
using namespace std;

// 计算字符串 s[0..n-1] 的前缀函数 (KMP)，复杂度 O(n)
// P[0]=0，对其他的 i，有最大的 P[i]，使得 s[0..P[i]-1] 等于 s[i-P[i]+1..i]
vector<int> calcP(const char *s, int n) {
    vector<int> P(n);
    for(int x=0,y=1; y<n; y++) {
        while(x && s[x]!=s[y]) x=P[x-1];
        if(s[x]==s[y]) P[y]=++x;
    }
    return P;
}

// 计算字符串 s[0..n-1] 的后缀函数 (Z-Function)，复杂度 O(n)，俗称扩展 KMP
// Z[0]=0，对其他的 i，有最大的 Z[i]，使得 s[0..Z[i]-1] 等于 s[i..i+Z[i]-1]
vector<int> calcZ(const char *s, int n) {
    vector<int> Z(n);
    for(int i=1,x=0,y=0; i<n; i++) {
        if(i<=y) Z[i]=min(y-i,Z[i-x]);
        while(i+Z[i]<n && s[i+Z[i]]==s[Z[i]]) Z[i]++;
        if(y<=i+Z[i]) x=i,y=i+Z[i];
    }
    return Z;
```

```
}
```

## 6.10 模式匹配 (kmp)

```
//模式匹配,kmp 算法, 复杂度 O(m+n)
//返回匹配位置,-1 表示匹配失败, 传入匹配串和模式串和长度
//可更改元素类型, 更换匹配函数
const int MAXN = 10000;
#define _match(a, b) ((a) == (b))

template <class elemType>
int patMatch(int ls, const elemType *str, int lp, const elemType *pat) {
    int fail[MAXN] = {-1}, i = 0, j;
    for (j = 1; j < lp; j++) {
        for (i = fail[j - 1]; i >= 0 && !_match(pat[i + 1], pat[j]); i = fail[i]);
        fail[j] = (_match(pat[i + 1], pat[j]) ? i + 1 :  - 1);
    }
    for (i = j = 0; i < ls && j < lp; i++) {
        if (_match(str[i], pat[j])) {
            j++;
        } else if (j) {
            j = fail[j - 1] + 1;
            i--;
        }
    }
    return j == lp ? (i - lp) : -1;
}

// 统计次数

#define MAXN 10000
#define _match(a,b) ((a)==(b))
typedef char elem_t;

int pat_match(int ls,elem_t *str,int lp,elem_t *pat) {
    int ret = 0;
    int fail[MAXN]= {-1},i=0,j;
    for (j=1; j<lp; j++) {
        for (i=fail[j-1]; i>=0&&!_match(pat[i+1],pat[j]); i=fail[i]);
        fail[j]=(_match(pat[i+1],pat[j])?i+1:-1);
    }
    for (i=j=0; i<ls; i++) {
        if (_match(str[i],pat[j])) {
            j++;
            if (j == lp) {
                ++ret;
                j=fail[j-1]+1;
            }
        } else if (j)
            j=fail[j-1]+1,i--;
    }
    return ret;
}
```

```
// 扩展 KMP, 复杂度 O(m+n)
// 传入匹配串 str 和模式串 pat 及长度, 返回 A[i] 值与 B[i] 值
// A[i] 表示 pat[i..m-1] 与 pat[0..m-1] 的最长公共前缀的长度
// B[i] 表示 str[i..n-1] 与 pat[0..m-1] 的最长公共前缀的长度

void extKMP(int n, const char str[], int m, const char pat[], int A[], int B[]) {
    A[0] = m;
    int ind = 0, k = 1;
    while (ind + 1 < m && pat[ind + 1] == pat[ind]) ind++;
    A[1] = ind;
    for (int i = 2; i < m; i++) {
        if (i <= k + A[k] - 1 && A[i - k] + i < k + A[k]) {
            A[i] = A[i - k];
        } else {
            ind = max(0, k + A[k] - i);
            while (ind + i < m && pat[ind + i] == pat[ind]) ind++;
            A[i] = ind, k = i;
        }
    }
    ind = 0, k = 0;
    while (ind < n && str[ind] == pat[ind]) ind++;
    B[0] = ind;
    for (int i = 1; i < n; i++) {
        if (i <= k + B[k] - 1 && A[i - k] + i < k + B[k]) {
            B[i] = A[i - k];
        } else {
            ind = max(0, k + B[k] - i);
            while (ind + i < n && ind < m && str[ind + i] == pat[ind]) ind++;
            B[i] = ind, k = i;
        }
    }
}
```

# Chapter 7

# 图论

## 7.1 NP 搜索

### 7.1.1 带权最大团

```
/* wclique.c exact algorithm for finding one maximum-weight
   clique in an arbitrary graph,
```

```
   10.2.2000, Patric R. J. Ostergard,
   patric.ostergard@hut.fi */

/* compile: gcc wclique.c -o wclique -O2 */

/* usage: wclique infile */

/* infile format: see http://www.tcs.hut.fi/~pat/wclique.html */

#include <stdio.h>
#include <sys/times.h>
#include <sys/types.h>

#define INT_SIZE (8*sizeof(int))
#define TRUE 1
#define FALSE 0
#define MAX_VERTEX 2000 /* maximum number of vertices */
#define MAX_WEIGHT 1000000 /* maximum weight of vertex */
#define is_edge(a,b) (bit[a][b/INT_SIZE]&(mask[b%INT_SIZE]))

int Vnbr,Enbr;          /* number of vertices/edges */
int clique[MAX_VERTEX]; /* table for pruning */
int bit[MAX_VERTEX][MAX_VERTEX/INT_SIZE+1];
int wt[MAX_VERTEX];

int pos[MAX_VERTEX];    /* reordering function */
int set[MAX_VERTEX];    /* current clique */
int rec[MAX_VERTEX];    /* best clique so far */
int record;     /* weight of best clique */
int rec_level;          /* # of vertices in best clique */

unsigned mask[INT_SIZE];
void graph();           /* reads graph */

struct tms bf;
int timer1;
double timer11;

main (argc,argv)
int argc;
char *argv[];
{
    int i,j,k,p;
    int min_wt,max_nwt,wth;
    int new[MAX_VERTEX],used[MAX_VERTEX];
    int nwt[MAX_VERTEX];
    int count;
    FILE *infile;

```

```
    /* read input */
    if(argc < 2) {
        printf("Usage: wclique infile\n");
        exit(1);
    }
    if((infile=fopen(argv[1],"r"))==NULL)
        fileerror();

    /* initialize mask */
    mask[0] = 1;
    for(i=1; i<INT_SIZE; i++)
        mask[i] = mask[i-1]<<1;

    /* read graph */
    graph(infile);

    /* "start clock" */
    times(&bf);
    timer1 = bf.tms_utime;

    /* order vertices */
    for(i=0; i<Vnbr; i++) {
        nwt[i] = 0;
        for(j=0; j<Vnbr; j++)
            if (is_edge(i,j)) nwt[i] += wt[j];
    }
    for(i=0; i<Vnbr; i++)
        used[i] = FALSE;
    count = 0;
    do {
        min_wt = MAX_WEIGHT+1;
        max_nwt = -1;
        for(i=Vnbr-1; i>=0; i--)
            if((!used[i])&&(wt[i]<min_wt))
                min_wt = wt[i];
        for(i=Vnbr-1; i>=0; i--) {
            if(used[i]||(wt[i]>min_wt)) continue;
            if(nwt[i]>max_nwt) {
                max_nwt = nwt[i];
                p = i;
            }
        }
        pos[count++] = p;
        used[p] = TRUE;
        for(j=0; j<Vnbr; j++)
            if((!used[j])&&(j!=p)&&(is_edge(p,j)))
                nwt[j] -= wt[p];
    } while(count<Vnbr);

    /* main routine */
    record = 0;
    wth = 0;
    for(i=0; i<Vnbr; i++) {
        wth += wt[pos[i]];
```

```
        sub(i,pos,0,0,wth);
        clique[pos[i]] = record;
        times(&bf);
        timer11 = (bf.tms_utime - timer1)/100.0;
        printf("level = %3d(%d) best = %2d time = %8.2f\n",i+1,Vnbr,record,timer11);
    }
    printf("Record: ");
    for(i=0; i<rec_level; i++)
        printf ("%d ",rec[i]);
    printf ("\n");
}

int sub(ct,table,level,weight,l_weight)
int ct,level,weight,l_weight;
int *table;
{
    register int i,j,k;
    int best;
    int curr_weight,left_weight;
    int newtable[MAX_VERTEX];
    int *p1,*p2;

    if(ct<=0) { /* 0 or 1 elements left; include these */
        if(ct==0) {
            set[level++] = table[0];
            weight += l_weight;
        }
        if(weight>record) {
            record = weight;
            rec_level = level;
            for (i=0; i<level; i++) rec[i] = set[i];
        }
        return 0;
    }
    for(i=ct; i>=0; i--) {
        if((level==0)&&(i<ct)) return 0;
        k = table[i];
        if((level>0)&&(clique[k]<=(record-weight))) return 0;  /* prune */
        set[level] = k;
        curr_weight = weight+wt[k];
        l_weight -= wt[k];
        if(l_weight<=(record-curr_weight)) return 0; /* prune */
        p1 = newtable;
        p2 = table;
        left_weight = 0;
        while (p2<table+i) {
            j = *p2++;
            if(is_edge(j,k)) {
                *p1++ = j;
                left_weight += wt[j];
            }
        }
        if(left_weight<=(record-curr_weight)) continue;
        sub(p1-newtable-1,newtable,level+1,curr_weight,left_weight);
    }
    return 0;
```

```
}

void graph(fp)
FILE *fp;
{
    register int i,j,k;
    int weight,degree,entry;

    if(!fscanf(fp,"%d %d\n",&Vnbr,&Enbr))
        fileerror();
    for(i=0; i<Vnbr; i++)    /* empty graph table */
        for(j=0; j<Vnbr/INT_SIZE+1; j++)
            bit[i][j] = 0;
    for(i=0; i<Vnbr; i++) {
        if(!fscanf(fp,"%d %d",&weight,&degree))
            fileerror();
        wt[i] = weight;
        for(j=0; j<degree; j++) {
            if(!fscanf(fp,"%d",&entry))
                fileerror();
            bit[i][entry/INT_SIZE] |= mask[entry%INT_SIZE]; /* record edge */
        }
    }
    fclose(fp);
}

int fileerror() {
    printf("Error in graph file\n");
    exit();
}
```

### 7.1.2 最大团 (n 小于 64)(faster)

```
/**
 * WishingBone's ACM/ICPC Routine Library
 *
 * maximum clique solver
 */

// 不知道怎么用……

#include <vector>

using std::vector;

// clique solver calculates both size and consitution of maximum clique
// uses bit operation to accelerate searching
// graph size limit is 63, the graph should be undirected
// can optimize to calculate on each component, and sort on vertex degrees
// can be used to solve maximum independent set
```

```cpp
class clique {
public:
    static const long long ONE = 1;
    static const long long MASK = (1 << 21) - 1;
    char *bits;
    int n, size, cmax[63];
    long long mask[63], cons;
    // initiate lookup table
    clique() {
        bits = new char[1 << 21];
        bits[0] = 0;
        for (int i = 1; i < 1 << 21; ++i) bits[i] = bits[i >> 1] + (i & 1);
    }
    ~clique() {
        delete bits;
    }
    // search routine
    bool search(int step, int size, long long more, long long con);
    // solve maximum clique and return size
    int sizeClique(vector<vector<int> > &mat);
    // solve maximum clique and return constitution
    vector<int> consClique(vector<vector<int> > &mat);
};

// search routine
// step is node id, size is current solution, more is available mask, cons is
// constitution mask
bool clique::search(int step, int size, long long more, long long cons) {
    if (step >= n) {
        // a new solution reached
        this->size = size;
        this->cons = cons;
        return true;
    }
    long long now = ONE << step;
    if ((now & more) > 0) {
        long long next = more & mask[step];
        if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >>
                42] >= this->size
                && size + cmax[step] > this->size) {
            // the current node is in the clique
            if (search(step + 1, size + 1, next, cons | now)) return true;
        }
    }
    long long next = more & ~now;
    if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >> 42]
            > this->size) {
        // the current node is not in the clique
        if (search(step + 1, size, next, cons)) return true;
    }
    return false;
```

```cpp
}

// solve maximum clique and return size
int clique::sizeClique(vector<vector<int> > &mat) {
    n = mat.size();
    // generate mask vectors
    for (int i = 0; i < n; ++i) {
        mask[i] = 0;
        for (int j = 0; j < n; ++j) if (mat[i][j] > 0) mask[i] |= ONE << j;
    }
    size = 0;
    for (int i = n - 1; i >= 0; --i) {
        search(i + 1, 1, mask[i], ONE << i);
        cmax[i] = size;
    }
    return size;
}

// solve maximum clique and return constitution
// calls sizeClique and restore cons
vector<int> clique::consClique(vector<vector<int> > &mat) {
    sizeClique(mat);
    vector<int> ret;
    for (int i = 0; i < n; ++i) if ((cons & (ONE << i)) > 0) ret.push_back(i);
    return ret;
}
```

### 7.1.3 最大团

```cpp
const int maxn = 50;

void clique(int n, int mat[][maxn], int num, int U[], int size, int C[], int &_max, int \
ok) {
    int i, j, k, tmp[maxn];
    if (num == 0) {
        if (size > _max) {
            ok = 1;
            _max = size;
        }
        return;
    }
    for (i = 0; i < num && !ok; ++i) {
        if (size + num  - i <= _max) return;
        if (size + C[U[i]] <= _max) return;
        for (k = 0, j = i + 1; j < num; ++j) if (mat[U[i]][U[j]])
                tmp[k++] = U[j];
        clique(n, mat, k, tmp, size + 1, C, _max, ok);
    }
}

int max_clique(int n, int mat[][maxn]) {
    int i, j, k, U[maxn], C[maxn], _max;
    for (_max = 0, i = n - 1; i >= 0; --i) {
        for (k = 0, j = i + 1; j < n; ++j) if (mat[i][j])
                U[k++] = j;
```

```
        clique(n, mat, k, U, 1, C, _max, 0);
        C[i] = _max;
    }
    return _max;
}
```

## 7.2 匹配

### 7.2.1 一般图匹配 (Blossom)

```
// 一般图最大基数匹配 (Gabow) By 猛犸也钻地 @ 2012.05.02

#include <vector>
#include <cstring>
using namespace std;

class Blossom {
public:
    static const int SIZE = 1005; // 最大结点个数
    int cnt,mate[SIZE]; // mate[] 为配偶结点的编号, 没有匹配上的点为 -1
    // 传入结点个数 n 及各结点的出边 e[], 返回匹配点对的数量 cnt
    int gao(int n, const vector<int> e[]) { // 复杂度 O(n^3)
        memset(mate,-1,sizeof(mate));
        for(int z=0; z<n; z++) if(mate[z]<0) {
                for(int i=0; i<n; i++) tag[i]=-2;
                int q[SIZE],push=1,pop=0;
                tag[q[0]=z]=-1;
                while(push!=pop) {
                    int x=q[pop++%SIZE];
                    for(size_t i=0; i<e[x].size(); i++) {
                        int y=e[x][i];
                        if(mate[y]<0 && z!=y) {
                            modify(mate[y]=x,y,n);
                            i=push=pop=1234567890;
                        } else if(tag[y]>=-1) {
                            memset(at,0,sizeof(at));
                            travel(x,n),travel(y,n);
                            for(int c=0; c<n; c++) if(at[c] && tag[c]<-1) {
                                    tag[c]=x+y*n+n;
                                    q[push++%SIZE]=c;
                                }
                        } else if(tag[mate[y]]<-1) {
                            tag[mate[y]]=x;
                            q[push++%SIZE]=mate[y];
                        }
                    }
                }
            }
        for(int i=cnt=0; i<n; i++) if(mate[i]>i) cnt++;
        return cnt;
    }
private:
    int at[SIZE],tag[SIZE];
```

```
    void modify(int x, int y, int n) {
        int z=mate[x];
        mate[x]=y;
        if(z<0 || mate[z]!=x) return;
        if(tag[x]<n) {
            mate[z]=tag[x];
            modify(mate[z],z,n);
        } else {
            y=tag[x]/n-1;
            z=tag[x]%n;
            modify(y,z,n);
            modify(z,y,n);
        }
    }
    void travel(int x, int n) {
        int tmp[SIZE];
        memcpy(tmp,mate,sizeof(tmp));
        modify(x,x,n);
        for(int i=0; i<n; i++)
            if(mate[i]!=tmp[i]) at[i]^=1,mate[i]=tmp[i];
    }
};
```

### 7.2.2 二分图最佳匹配 (kuhn munkras 邻接阵形式)

```
//二分图最佳匹配,kuhn munkras 算法, 邻接阵形式, 复杂度 O(m*m*n)
//返回最佳匹配值, 传入二分图大小 m,n 和邻接阵 mat, 表示权值
//match1,match2 返回一个最佳匹配, 未匹配顶点 match 值为 -1
//一定注意 m<=n, 否则循环无法终止
//最小权匹配可将权值取相反数
#include <cstring>
const int MAXN = 310;
const int INF = 1000000000;
#define _clr(x) memset(x, 0xff, sizeof(int) * n)

int kuhnMunkras(int m, int n, int mat[][MAXN], int *match1, int *match2) {
    int s[MAXN + 1], t[MAXN], l1[MAXN], l2[MAXN], p, q, ret = 0, i, j, k;
    for (i = 0; i < m; i++) {
        l1[i] =  - INF;
        for (j = 0; j < n; j++) {
            l1[i] = mat[i][j] > l1[i] ? mat[i][j]: l1[i];
        }
    }
    for (i = 0; i < n; l2[i++] = 0);
    _clr(match1);
    _clr(match2);
    for (i = 0; i < m; i++) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            k = s[p];
            for (j = 0; j < n && match1[i] < 0; j++) {
                if (l1[k] + l2[j] == mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
```

```
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
        if (match1[i] < 0) {
            i--;
            p = INF;
            for (k = 0; k <= q; k++) {
                for (j = 0; j < n; j++) {
                    if (t[j] < 0 && l1[s[k]] + l2[j] - mat[s[k]][j] < p) {
                        p = l1[s[k]] + l2[j] - mat[s[k]][j];
                    }
                }
            }
            for (j = 0; j < n; j++) {
                l2[j] += t[j] < 0 ? 0 : p;
            }
            for (k = 0; k <= q; k++) {
                l1[s[k]] -= p;
            }
        }
    }
    for (i = 0; i < m; i++) {
        ret += mat[i][match1[i]];
    }
    return ret;
}
```

### 7.2.3 二分图最佳匹配 (kuhn munkras 邻接阵形式)yxdb

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2005;
const int INF = 1e9 + 7;

struct KuhnMunkres {
    int sum, mx[N], my[N], lx[N], ly[N], sx[N], sy[N];
    bool vx[N], vy[N];
    int gao(int n, int e[N][N]) {
        fill_n(mx, n, -1);
        fill_n(my, n, -1);
        fill_n(ly, n, 0);
        for (int i = 0; i < n; ++i) lx[i] = *max_element(e[i], e[i]+n);
        for (int x = 0, y, z, w; x < n; ++x) {
            fill_n(vx, n, false);
            fill_n(vy, n, false);
            queue<int> q;
            vx[x] = true;
            for (int i = 0; i < n; ++i) {
                sx[i] = x, sy[i] = lx[x] + ly[i] - e[x][i];
```

```cpp
                if (!sy[i]) q.push(i), vy[i] = true;
            }
            while (1) {
                if (q.empty()) {
                    int delta = *min_element(sy, sy+n);
                    for (int i = 0; i < n; ++i) {
                        if (vx[i]) lx[i] -= delta;
                        if (vy[i]) ly[i] += delta;
                        else if (!(sy[i] -= delta)) q.push(i), vy[i] = true;
                    }
                } else {
                    y = q.front();
                    q.pop();
                    if (!~my[y]) break;
                    vx[z = my[y]] = vy[y] = true, sy[y] = INF;
                    for (int i = 0; i < n; ++i) if (!vy[i]) {
                            int d = lx[z] + ly[i] - e[z][i];
                            if (sy[i] > d) sy[i] = d, sx[i] = z;
                            if (!sy[i]) q.push(i), vy[i] = true;
                        }
                }
            }
            for (; ~y; y = w) z = sx[y], w = mx[z], mx[z] = y, my[y] = z;
        }
        for (int i = sum = 0; i < n; ++i) sum += lx[i] + ly[i];
        return sum;
    }
} km;

int n, e[N][N];

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j) e[i][j] = (i+1)*(j+1);
    int ans = km.gao(n, e);
    for (int i = 0; i < n; ++i) {
        assert(km.mx[i] == i && km.my[i] == i);
        ans -= (i+1)*(i+1);
    }
    assert(!ans);
}
```

### 7.2.4 二分图最大匹配 (hopcroft kart 邻接表形式)

```cpp
// Hopcroft_Karp matching algorithm, 图的大小为 n 和 m, 返回最大匹配数
// vector 存储, 复杂度 O(sqrt(V)*E)
// match1 和 match2 为最大匹配, 未匹配节点 match1 为 -1, match2 为 n
// 每次使用之前将边信息存入 E

#include <cstring>
#include <vector>
using namespace std;

const int MAXN = 50005;
const int MAXM = 200005;
```

```
int n, m;
int match1[MAXN], match2[MAXN];
int Q[MAXN], D1[MAXN], D2[MAXN];
vector <int> E[MAXN];

inline bool bfs() {
    int s = 0, t = 0, u, v;
    memset(D1, -1, sizeof(D1));
    memset(D2, -1, sizeof(D2));
    for (int i = 0; i < n; i++)
        if (match1[i] == -1)
            Q[t++] = i, D1[i] = 0;
    while (s != t)
        if ((u = Q[s++]) != n)
            for (int i = 0; i < (int)E[u].size(); i++)
                if (D2[v = E[u][i]] == -1) {
                    D2[v] = D1[u] + 1;
                    if (D1[match2[v]] == -1)
                        D1[Q[t++] = match2[v]] = D2[v] + 1;
                }
    return D1[n] != -1;
}

bool dfs(int u) {
    for (int i = 0, v; i < (int)E[u].size(); i++)
        if (D2[v = E[u][i]] == D1[u] + 1 && (D2[v] = -1) && (match2[v] == n || dfs(match\
2[v]))) {
            match1[u] = v;
            match2[v] = u;
            D1[u] = -1;
            return true;
        }
    D1[u] = -1;
    return false;
}

inline int hopcroft_karp() {
    memset(match1, -1, sizeof(match1));
    for (int i = 0; i < m; i++)
        match2[i] = n;
    int ret = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0, u; j < E[i].size(); j++)
            if (match2[u = E[i][j]] == n) {
                match1[i] = u;
                match2[u] = i;
                ret++;
                break;
            }
    while (bfs())
        for (int i = 0; i < n; i++)
            if (match1[i] == -1 && dfs(i))
                ret++;
    return ret;
}
```

### 7.2.5 二分图最大匹配 (hungary bfs 邻接阵形式)

```
//二分图最大匹配,hungary 算法, 邻接阵形式, 复杂度 O(n*n*m)
//返回最大匹配数, 传入二分图大小 n,m 和邻接阵 mat, 非零元素表示有边
//match1,match2 返回一个最大匹配, 未匹配顶点 match 值为 -1
#include <cstring>
const int MAXN = 310;
#define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)

int hungary(int n, int m, const bool mat[][MAXN], int *match1, int *match2) {
    int s[MAXN + 1], t[MAXN], p, q, ret = 0, i, j, k;
    _clr(match1);
    _clr(match2);
    for (i = 0; i < n; ret += (match1[i++] >= 0)) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            k = s[p];
            for (j = 0; j < m && match1[i] < 0; j++) {
                if (mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
    }
    return ret;
}
```

### 7.2.6 二分图最大匹配 (hungary dfs 邻接阵形式)

```
//二分图最大匹配,hungary dfs 算法, 邻接阵形式, 复杂度 O(n*n*m)
//返回最大匹配数, 传入二分图大小 n,m 和邻接阵 mat, 非零元素表示有边
//match1,match2 返回一个最大匹配, 未匹配顶点 match 值为 -1
#define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)
const int MAXN = 200;
int mk[MAXN], match1[MAXN], match2[MAXN], mat[MAXN][MAXN], n, m;
int path(int i) {
    for (int j = 0; j < m; j++)
        if (mat[i][j] && !(mk[j]++) && (match2[j] < 0 || path(match2[j]))) {
            match1[i] = j;
            match2[j] = i;
            return 1;
        }
    return 0;
}
int hungary() {
    int res(0);
```

```
    _clr(match1);
    _clr(match2);
    for (int i = 0; i < n; i++)
        if (match1[i] < 0) {
            memset(mk, 0, sizeof(mk));
            res += path(i);
        }
    return res;
}
```

## 7.3 应用

### 7.3.1 2-sat

```
// O(N + M)
// 调用 init() 初始化, 传入节点个数, 节点标号必须从 0 开始
// 调用 add_edge() 添加边, 调用 add_var() 添加变量的初始值
// 调用 solve() 求解 2sat 如果有解存在 mark 中
// 对于变量 x, x*2 表示 x, x * 2 + 1 表示 ~x, mark[x << 1] 表示 x 的值
struct TwoSAT {
    static const int MAXN = 200000 + 10;
    vector<int> G[MAXN], SCC[MAXN];
    int low[MAXN], dfn[MAXN], stk[MAXN];
    int col[MAXN], mark[MAXN];
    int scc_cnt, top, sz, n;
    void init(int n) {
        this->n = n;
        scc_cnt = 0;
        for (int i = 0; i < n * 2; ++ i) {
            G[i].clear();
            SCC[i].clear();
        }
    }
    void dfs(int x) {
        low[x] = dfn[x] = ++ sz;
        stk[top ++] = x;
        mark[x] = true;
        for (int i = 0, y; i < (int)G[x].size(); ++ i) {
            if (!dfn[y = G[x][i]]) {
                dfs(y);
                low[x] = min(low[x], low[y]);
            } else if (mark[y]) low[x] = min(low[x], dfn[y]);
        }
        if (dfn[x] == low[x]) {
            SCC[scc_cnt ++].clear();
            for (int y; ; ) {
                mark[y = stk[-- top]] = false;
                SCC[scc_cnt - 1].push_back(y);
                col[y] = scc_cnt - 1;
                if (y == x) break;
            }
        }
    }
    inline int get_val(int x) {
```

```
        int r = (x & 1) ? x ^ 1 : x;
        if (mark[r] == -1) return -1;
        return (x & 1) ? !mark[r] : mark[r];
    }
    void construct() {
        for (int i = 0; i < n * 2; ++ i) mark[i] = -1;
        for (int i = 0, j, val; i < scc_cnt; ++ i) {
            for (val = 1, j = 0; j < (int)SCC[i].size(); ++ j) {
                int cur = SCC[i][j];
                if (get_val(cur) == 0) val = 0;
                for (int k = 0; k < (int)G[cur].size(); ++ k)
                    if (get_val(G[cur][k]) == 0) val = 0;
                if (val == 0) break;
            }
            for (j = 0; j < (int)SCC[i].size(); ++ j) {
                if (SCC[i][j] & 1) mark[SCC[i][j] ^ 1] = !val;
                else mark[SCC[i][j]] = val;
            }
        }
    }
    bool solve() {
        for (int i = 0; i < 2 * n; ++ i) mark[i] = false, dfn[i] = 0;
        top = sz = 0;
        for (int i = 0; i < 2 * n; ++ i)
            if (!dfn[i]) dfs(i);
        for (int i = 0; i < 2 * n; i += 2)
            if (col[i] == col[i ^ 1]) return false;
        construct();
        return true;
    }
    void add_edge(int x, int y) { // x -> y, 0 <= x, y < 2 * n
        G[x].push_back(y);
    }
    void add_var(int x, int xv) { // x = xv, 0 <= x < n
        x = x << 1 | xv;
        G[x ^ 1].push_back(x);
    }
};
```

### 7.3.2 前序表转化

```
//将用边表示的树转化为前序表示的树
//传入节点数 n 和邻接表 list[], 邻接表必须是双向的, 会在函数中释放
//pre[] 返回前序表,map[] 返回前序表中的节点到原来节点的映射
const int MAXN = 10000;
struct Node {
    int to;
    Node *next;
};

void prenode(int n, Node *list[], int *pre, int *map, int *v, int now, int last, int &id\
) {
    Node *t;
    int p = id++;
```

```
    for (v[map[p] = now] = 1, pre[p] = last; list[now];) {
        t = list[now];
        list[now] = t->next;
        if (!v[t->to]) {
            prenode(n, list, pre, map, v, t->to, p, id);
        }
    }
}

void makepre(int n, Node *list[], int *pre, int *map) {
    int v[MAXN], id = 0, i;
    for (i = 0; i < n; v[i++] = 0);
    prenode(n, list, pre, map, v, 0,  - 1, id);
}
```

### 7.3.3 拓扑排序 (邻接阵形式)

```
//拓扑排序, 邻接阵形式, 复杂度 O(n^2)
//如果无法完成排序, 返回 0, 否则返回 1,ret 返回有序点列
//传入图的大小 n 和邻接阵 mat, 不相邻点边权 0
const int MAXN = 100;

bool toposort(int n, int mat[][MAXN], int *ret) {
    int d[MAXN], i, j, k;
    for (i = 0; i < n; i++) {
        for (d[i] = j = 0; j < n; d[i] += mat[j++][i]);
    }
    for (k = 0; k < n; ret[k++] = i) {
        for (i = 0; d[i] && i < n; i++);
        if (i == n) {
            return false;
        }
        for (d[i] =  - 1, j = 0; j < n; j++) {
            d[j] -= mat[i][j];
        }
    }
    return true;
}
```

### 7.3.4 无向图全局最小割

```
// 无向图全局最小割 (Stoer-Wagner) By 猛犸也钻地 @ 2012.08.22

#include <vector>
#include <algorithm>
using namespace std;

class StoerWagner {
public:
    typedef int VAL; // 权值的类型
    static const int SIZE = 505;        // 最大结点个数
    static const VAL INF = 1000000007;  // 最大权值之和
    VAL sum,e[SIZE][SIZE];
```

```
    // 传入结点个数 n 及权值矩阵 a[][]，返回无向图全局最小割的边权之和 sum
    // 对于矩阵 a[][] 中不存在的边，权值设为 0
    int gao(int n, const VAL a[SIZE][SIZE]) {
        vector<int> v,idx(n);
        for(int i=0; i<n; i++) copy(a[i],a[i]+n,e[idx[i]=i]);
        for(int i=0; i<n; i++) e[i][i]=0;
        for(sum=INF; idx.size()>=2; n=idx.size()) {
            vector<VAL> s(n);
            for(int i=0; i<n; i++) v.push_back(i);
            int p=0,t=0;
            while(v.size()) {
                int m=v.size(),x=-1;
                for(int i=0; i<m; i++) if(x<0 || s[x]<s[i]) x=i;
                for(int i=0; i<m; i++) s[i]+=e[idx[v[x]]][idx[v[i]]];
                v.erase(v.begin()+x);
                s.erase(s.begin()+x);
                swap(p=v[x],t);
            }
            VAL now=0;
            for(int i=0; i<n; i++) if(i!=t) now+=e[idx[t]][idx[i]];
            for(int i=0; i<n; i++) {
                e[idx[i]][idx[p]]+=e[idx[i]][idx[t]];
                e[idx[p]][idx[i]]+=e[idx[t]][idx[i]];
            }
            idx.erase(idx.begin()+t);
            sum=min(sum,now);
        }
        return sum;
    }
};
```

### 7.3.5 无向图最小环

```
// 无向图最小环 (Floyd) By 猛犸也钻地 @ 2012.09.13

#include <vector>
#include <cstring>
using namespace std;

class Floyd {
public:
    typedef int VAL; // 权值的类型
    static const int SIZE = 105;
    vector<int> path;
    VAL len[SIZE][SIZE],ans;
    int src[SIZE][SIZE];
    // 传入结点个数 n 及权值矩阵 a[][]，返回最小环的长度 ans，方案记在 path 中
    // 对于矩阵 a[][] 中不存在的边，权值设为 1e9+7 或 0x7F7F7F7F 之类的极大值
    VAL gao(int n, const VAL a[SIZE][SIZE]) {
        ans=1e9+7; // 若最后的返回值大于等于 1e9，则不存在最小环
        memset(src,-1,sizeof(src));
        memcpy(len, a,sizeof(len));
        for(int k=0; k<n; k++) {
```

```
            for(int i=0; i<k; i++) for(int j=i+1; j<k; j++) {
                    VAL tmp=a[k][i]+a[j][k];
                    if(len[i][j]>=ans-tmp) continue;
                    path.clear();
                    getpath(i,j);
                    path.push_back(k);
                    path.push_back(i);
                    ans=tmp+len[i][j];
                }
            for(int i=0; i<n; i++) for(int j=0; j<n; j++) {
                    VAL tmp=len[i][k]+len[k][j];
                    if(tmp>=len[i][j]) continue;
                    len[i][j]=tmp;
                    src[i][j]=k;
                }
        }
        return ans;
    }
private:
    void getpath(int i, int j) {
        int k=src[i][j];
        if(~k) {
            getpath(i,k);
            getpath(k,j);
        } else {
            path.push_back(j);
        }
    }
};
```

### 7.3.6 最佳边割集

```
//最佳边割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i]: c[j];
```

```
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int bestEdgeCut(int n, int mat[][MAXN], int source, int sink, int set[][2], int &mincost\
) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;
    if (source == sink) {
        return -1;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m0[i][j] = mat[i][j];
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    mincost = last = maxFlow(n, m, source, sink);
    for (k = 0; k < n && last; k++) {
        for (l = 0; l < n && last; l++) {
            if (m0[k][l]) {
                for (i = 0; i < n + n; i++) {
                    for (j = 0; j < n + n; j++) {
                        m[i][j] = m0[i][j];
                    }
                }
                m[k][l] = 0;
                if (maxFlow(n, m, source, sink) == last - mat[k][l]) {
                    set[ret][0] = k;
                    set[ret++][1] = l;
                    m0[k][l] = 0;
                    last -= mat[k][l];
                }
            }
        }
    }
    return ret;
}
```

### 7.3.7 最佳顶点割集

```
//最佳顶点割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
```

```
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int bestVertexCut(int n, int mat[][MAXN], int *cost, int source, int sink, int *set, int\
 &mincost) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;
    if (source == sink || mat[source][sink]) {
        return -1;
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m0[i][j] = 0;
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j]) {
                m0[i][n + j] = INF;
            }
        }
    }
    for (i = 0; i < n; i++) {
        m0[n + i][i] = cost[i];
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    mincost = last = maxFlow(n + n, m, source, n + sink);
    for (k = 0; k < n && last; k++) {
```

```
        if (k != source && k != sink) {
            for (i = 0; i < n + n; i++) {
                for (j = 0; j < n + n; j++) {
                    m[i][j] = m0[i][j];
                }
            }
            m[n + k][k] = 0;
            if (maxFlow(n + n, m, source, n + sink) == last - cost[k]) {
                set[ret++] = k;
                m0[n + k][k] = 0;
                last -= cost[k];
            }
        }
    }
    return ret;
}
```

### 7.3.8 最小路径覆盖

```
//最小路径覆盖,O(n^3)
//求解最小的路径覆盖图中所有点, 有向图无向图均适用
//注意此问题等价二分图最大匹配, 可以用邻接表或正向表减小复杂度
//返回最小路径条数,pre 返回前指针 (起点 -1),next 返回后指针 (终点 -1)
#include <cstring>
const int MAXN = 310;
#define _clr(x) memset(x, 0xff, sizeof(int) * n)

int hungary(int n, const bool mat[][MAXN], int *match1, int *match2) {
    int s[MAXN], t[MAXN], p, q, ret = 0, i, j, k;
    _clr(match1);
    _clr(match2);
    for (i = 0; i < n; ret += (match1[i++] >= 0)) {
        _clr(t);
        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
            for (k = s[p], j = 0; j < n && match1[i] < 0; j++) {
                if (mat[k][j] && t[j] < 0) {
                    s[++q] = match2[j];
                    t[j] = k;
                    if (s[q] < 0) {
                        for (p = j; p >= 0; j = p) {
                            match2[j] = k = t[j];
                            p = match1[k];
                            match1[k] = j;
                        }
                    }
                }
            }
        }
    }
    return ret;
}

inline int pathCover(int n, const bool mat[][MAXN], int *pre, int *next) {
    return n - hungary(n, mat, next, pre);
}
```

### 7.3.9 最小边割集

```cpp
//最小边割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int minEdgeCut(int n, int mat[][MAXN], int source, int sink, int set[][2]) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;
    if (source == sink) {
        return  - 1;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m0[i][j] = (mat[i][j] != 0);
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    last = maxFlow(n, m, source, sink);
    for (k = 0; k < n && last; k++) {
        for (l = 0; l < n && last; l++) {
            if (m0[k][l]) {
                for (i = 0; i < n + n; i++) {
```

```
                    for (j = 0; j < n + n; j++) {
                        m[i][j] = m0[i][j];
                    }
                }
                m[k][l] = 0;
                if (maxFlow(n, m, source, sink) < last) {
                    set[ret][0] = k;
                    set[ret++][1] = l;
                    m0[k][l] = 0;
                    last--;
                }
            }
        }
    }
    return ret;
}
```

### 7.3.10 最小顶点割集

```
//最小顶点割集
const int MAXN = 100;
const int INF = 1000000000;

int maxFlow(int n, int mat[][MAXN], int source, int sink) {
    int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
    while (true) {
        for (i = 0; i < n; i++) {
            v[i] = c[i] = 0;
        }
        for (c[source] = INF;;) {
            for (j = -1, i = 0; i < n; i++) {
                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
                    j = i;
                }
            }
            if (j < 0) {
                return ret;
            }
            if (j == sink) {
                break;
            }
            for (v[j] = 1, i = 0; i < n; i++) {
                if (mat[j][i] > c[i] && c[j] > c[i]) {
                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
                    p[i] = j;
                }
            }
        }
        for (ret += j = c[i = sink]; i != source; i = p[i]) {
            mat[p[i]][i] -= j;
            mat[i][p[i]] += j;
        }
    }
}

int minVertexCut(int n, int mat[][MAXN], int source, int sink, int *set) {
    int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;
```

```
    if (source == sink || mat[source][sink]) {
        return  - 1;
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m0[i][j] = 0;
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (mat[i][j]) {
                m0[i][n + j] = INF;
            }
        }
    }
    for (i = 0; i < n; i++) {
        m0[n + i][i] = 1;
    }
    for (i = 0; i < n + n; i++) {
        for (j = 0; j < n + n; j++) {
            m[i][j] = m0[i][j];
        }
    }
    last = maxFlow(n + n, m, source, n + sink);
    for (k = 0; k < n && last; k++) {
        if (k != source && k != sink) {
            for (i = 0; i < n + n; i++) {
                for (j = 0; j < n + n; j++) {
                    m[i][j] = m0[i][j];
                }
            }
            m[n + k][k] = 0;
            if (maxFlow(n + n, m, source, n + sink) < last) {
                set[ret++] = k;
                m0[n + k][k] = 0;
                last--;
            }
        }
    }
    return ret;
}
```

### 7.3.11 树的优化算法

```
const int MAXN = 1000;

//最大顶点独立集
int maxNodeIndependent(int n, int *pre, int *set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i]) {
            set[i] = 1;
            if (pre[i] != -1) {
                c[pre[i]] = 1;
```

```
            }
            ret++;
        }
    }
    return ret;
}

//最大边独立集
int maxEdgeIndependent(int n, int *pre, int *set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i] && pre[i] != -1 && !c[pre[i]]) {
            set[i] = 1;
            c[pre[i]] = 1;
            ret++;
        }
    }
    return ret;
}

//最小顶点覆盖集
int minNodeCover(int n, int *pre, int *set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i] && pre[i] != -1 && !c[pre[i]]) {
            set[i] = 1;
            c[pre[i]] = 1;
            ret++;
        }
    }
    return ret;
}

//最小顶点支配集
int minNodeDominant(int n, int *pre, int *set) {
    int c[MAXN], i, ret = 0;
    for (i = 0; i < n; i++) {
        c[i] = set[i] = 0;
    }
    for (i = n - 1; i >= 0; i--) {
        if (!c[i] && (pre[i] == -1 || !set[pre[i]])) {
            if (pre[i] !=  - 1) {
                set[pre[i]] = 1;
                c[pre[i]] = 1;
                if (pre[pre[i]] != -1) {
                    c[pre[pre[i]]] = 1;
                }
            } else {
                set[i] = 1;
            }
```

```
            ret++;
        }
    }
    return ret;
}
```

### 7.3.12 欧拉回路 (邻接阵形式)

```
//求欧拉回路或欧拉路, 邻接阵形式, 复杂度 O(n^2)
//返回路径长度,path 返回路径 (有向图时得到的是反向路径)
//传入图的大小 n 和邻接阵 mat, 不相邻点边权 0
//可以有自环与重边, 分为无向图和有向图

const int MAXN = 100;

void findPathU(int n, int mat[][MAXN], int now, int &step, int *path) {
    int i;
    for (i = n - 1; i >= 0; i--) {
        while (mat[now][i]) {
            mat[now][i]--;
            mat[i][now]--;
            findPathU(n, mat, i, step, path);
        }
    }
    path[step++] = now;
}

void findPathD(int n, int mat[][MAXN], int now, int &step, int *path) {
    int i;
    for (i = n - 1; i >= 0; i--) {
        while (mat[now][i]) {
            mat[now][i]--;
            findPathD(n, mat, i, step, path);
        }
    }
    path[step++] = now;
}

int euclidPath(int n, int mat[][MAXN], int start, int *path) {
    int ret = 0;
    findPathU(n, mat, start, ret, path);
//  findPathD(n, mat, start, ret, path);
    return ret;
}
```

## 7.4 生成树

### 7.4.1 多源最小树形图 (邻接阵形式)

```
//多源最小树形图,edmonds 算法, 邻接阵形式, 复杂度 O(n^3)
//返回最小生成树的长度, 构造失败返回负值
```

```
//传入图的大小 n 和邻接阵 mat, 不相邻点边权 inf
//可更改边权的类型,pre[] 返回树的构造, 用父结点表示
//传入时 pre[] 数组清零, 用 -1 标出可能的源点
#include <string.h>
#define MAXN 120
#define inf 1000000000
typedef int elem_t;
elem_t edmonds(int n,elem_t mat[][MAXN*2],int *pre) {
    elem_t ret=0;
    int c[MAXN*2][MAXN*2],l[MAXN*2],p[MAXN*2],m=n,t,i,j,k;
    for (i=0; i<n; l[i]=i,i++);
    do {
        memset(c,0,sizeof(c)),memset(p,0xff,sizeof(p));
        for (t=m,i=0; i<m; c[i][i]=1,i++);
        for (i=0; i<t; i++)
            if (l[i]==i&&pre[i]!=-1) {
                for (j=0; j<m; j++)
                    if
                    (l[j]==j&&i!=j&&mat[j][i]<inf&&(p[i]==-1||mat[j][i]<mat[p[i]][i]))
                        p[i]=j;
                if ((pre[i]=p[i])==-1)
                    return -1;
                if (c[i][p[i]]) {
                    for (j=0; j<=m; mat[j][m]=mat[m][j]=inf,j++);
                    for (k=i; l[k]!=m; l[k]=m,k=p[k])
                        for (j=0; j<m; j++)
                            if (l[j]==j) {
                                if (mat[j][k]-mat[p[k]][k]<mat[j][m])
                                    mat[j][m]=mat[j][k]-mat[p[k]][k];
                                if (mat[k][j]<mat[m][j])
                                    mat[m][j]=mat[k][j];
                            }
                    c[m][m]=1,l[m]=m,m++;
                }
                for (j=0; j<m; j++)
                    if (c[i][j])
                        for (k=p[i]; k!=-1&&l[k]==k; c[k][j]=1,k=p[k]);
            }
    } while (t<m);
    for (; m-->n; pre[k]=pre[m])
        for (i=0; i<m; i++)
            if (l[i]==m) {
                for (j=0; j<m; j++)
                    if (pre[j]==m&&mat[i][j]==mat[m][j])
                        pre[j]=i;
                if (mat[pre[m]][m]==mat[pre[m]][i]-mat[pre[i]][i])
                    k=i;
            }
    for (i=0; i<n; i++)
        if (pre[i]!=-1)
            ret+=mat[pre[i]][i];
    return ret;
}
```

### 7.4.2 最小生成树 (kruskal 邻接表形式)

```
//无向图最小生成树,kruskal 算法, 邻接表形式, 复杂度 O(mlogm)
//返回最小生成树的长度, 传入图的大小 n 和邻接表 list
//可更改边权的类型,edge[][2] 返回树的构造, 用边集表示
//如果图不连通, 则对各连通分支构造最小生成树, 返回总长度
#include <cstring>
const int MAXN = 200;
const int INF = 1000000000;

#define _run(x) for(; p[t = x]; x = p[x], p[t] = (p[x] ? p[x] : x))
#define _run_both _run(i); _run(j)

class DSet {
public:
    int p[MAXN], t;
    void init() {
        memset(p, 0, sizeof(p));
    }
    void setFriend(int i, int j) {
        _run_both;
        p[i] = (i == j ? 0 : j);
    }
    bool isFriend(int i, int j) {
        _run_both;
        return i == j && i;
    }
};

typedef double elemType;

struct Edge {
    int from, to;
    elemType len;
    Edge *next;
};

struct HeapNode {
    int a, b;
    elemType len;
};

#define _cp(a,b) ((a).len < (b).len)

class MinHeap {
public:
    HeapNode h[MAXN *MAXN];
    int n, p, c;
    void init() {
        n = 0;
    }
    void ins(HeapNode e) {
        for (p = ++n; p > 1 && _cp(e, h[p >> 1]); h[p] = h[p >> 1], p >>= 1);
        h[p] = e;
    }
    bool del(HeapNode &e) {
```

```
        if (!n) {
            return false;
        }
        e = h[p = 1];
        for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]); c <<\
= 1) {
            h[p] = h[c];
            p = c;
        }
        h[p] = h[n--];
        return true;
    }
};

elemType kruskal(int n, const Edge *list[], int edge[][2]) {
    DSet u;
    MinHeap h;
    const Edge *t;
    HeapNode e;
    elemType ret = 0;
    int i, m = 0;
    u.init(), h.init();
    for (i = 0; i < n; i++) {
        for (t = list[i]; t; t = t->next) {
            if (i < t->to) {
                e.a = i;
                e.b = t->to;
                e.len = t->len;
                h.ins(e);
            }
        }
    }
    while (m < n - 1 && h.del(e)) {
        if (!u.isFriend(e.a + 1, e.b + 1)) {
            edge[m][0] = e.a;
            edge[m][1] = e.b;
            ret += e.len;
            u.setFriend(e.a + 1, e.b + 1);
        }
    }
    return ret;
}
```

### 7.4.3 最小生成树 (prim+priority queue 邻接阵形式)

```
// 不是太苛刻的情况下推荐使用
// 复杂度为 O(|E|+|V|lg|V|), 但常数较大, 而且复杂度不是很严格
// 边权非负!

#define Rec pair<T, int>

template<class T>
T Prim(int n, vector<pair<int, T> > e[], T mind[], int *pre = NULL) {
    int s = 0;
    priority_queue<Rec, vector<Rec>, greater<Rec> > q;
```

```
    vector<bool> mark(n, false);

    // pre 为 NULL 则不做记录
    if (pre != NULL) {
        fill(pre, pre + n, -1);
        pre[s] = s;
    }
    // mind 初始化部分注意修改
    fill(mind, mind + n, numeric_limits<T>::max());
    mind[s] = T();
    q.push(make_pair(T(), s));

    T ret = T();

    while (!q.empty()) {
        s = q.top().second;
        if (!mark[s]) {
            mark[s] = true;
            ret += q.top().first;
            q.pop();
            for (typename vector<pair<int, T> >::const_iterator i = e[s].begin(); i != e\
[s].end(); ++i) {
                if (!mark[i->first] && mind[i->first] > i->second) {
                    mind[i->first] = i->second;
                    if (pre != NULL) {
                        pre[i->first] = s;
                    }
                    q.push(make_pair(mind[i->first], i->first));
                }
            }
        } else {
            q.pop();
        }
    }

    return ret;
}
```

### 7.4.4 最小生成树 (prim 邻接阵形式)

```
//无向图最小生成树,prim 算法, 邻接阵形式, 复杂度 O(n^2)
//返回最小生成树的长度, 传入图的大小 n 和邻接阵 mat, 不相邻点边权 INF
//可更改边权的类型,pre[] 返回树的构造, 用父结点表示, 根节点 (第一个)pre 值为 -1
//必须保证图的连通的!
const int MAXN = 200;
const int INF = 1000000000;

template <class elemType>
elemType prim(int n, const elemType mat[][MAXN], int *pre) {
    elemType mind[MAXN], ret = 0;
    int v[MAXN], i, j, k;
    for (i = 0; i<n; i++) {
        mind[i] = INF;
        v[i] = 0;
```

```
        pre[i] = -1;
    }
    for (mind[j = 0] = 0; j<n; j++) {
        for (k = -1, i = 0; i<n; i++) {
            if (!v[i] && (k == -1 || mind[i]<mind[k])) {
                k = i;
            }
        }
        v[k] = 1;
        ret += mind[k];
        for (i = 0; i < n; i++) {
            if (!v[i] && mat[k][i] < mind[i]) {
                mind[i] = mat[pre[i] = k][i];
            }
        }
    }
    return ret;
}
```

### 7.4.5 次最小生成树

```
// 次小生成树, 复杂度 O(n^2)
// 传入邻接阵 mat, 不存在边权 inf
// 返回次小生成树长度和树的构造 pre[]
// 如返回 inf 则不存在次小生成树
// 必须保证图的连通
const int maxn = 100;
const int inf = 1000000000;
typedef int elem_t;

elem_t prim(int n, elem_t mat[][maxn], int *pre) {
    elem_t min[maxn], ret = 0;
    int v[maxn], i, j, k;

    for (i = 0; i < n; ++i)
        min[i] = inf, v[i] = 0, pre[i] = -1;
    for (min[j = 0] = 0; j < n; ++j) {
        for (k = -1, i = 0; i < n; ++i) if (!v[i] && (k == -1 || min[i] < min[k]))
                k = i;
        for (v[k] = 1, ret += min[k], i = 0; i < n; ++i)
            if (!v[i] && mat[k][i] < min[i])
                min[i] = mat[pre[i] = k][i];
    }

    return ret;
}

elem_t sbmst(int n, elem_t mat[][maxn], int *pre) {
    elem_t min = inf, t, ret = prim(n, mat, pre);
    int i, j, ti, tj;
    for (i = 0; i < n; ++i) for (j = 0; j < n && pre[i] != -1; ++j) if (i != j && pre[i]\
 != j && pre[j] != i)
                if (mat[j][i] < inf && (t = mat[j][i] - mat[pre[i]][i]) < min)
                    min = t, ti = i, tj = j;
```

```
    pre[ti] = tj;
    return ret + min;
}
```

## 7.5 网络流

### 7.5.1 最大流 (dinic d)

```
/*
    最大流 Dinic 算法 by dd_engi
    1. 算法被封装成了一个 struct。
    2.struct 需要在全局变量中声明或者 new 出来，千万不要声明成栈上的局部变量。
    3. 每次使用前，dinic.init(S,T) 给定源与汇的编号，然后用 dinic.add_edge(x,y,w) 添加每条有 \
容量的边。
    4. 调用 dinic.flow() 进行计算，返回最大流的值；每条边的流量有储存在 edge.f 里。
    5. 同一个 struct 可以处理多组数据，但每次都要先 init。
    6. 不需要知道总点数，点的编号可以不连续，但是所有的编号都需要在 [0,MAXN) 之间。
    7. 可处理多重边。
    8.dinic.cut() 是一个附送的功能，调用 flow() 后，可用它求出最小割中的 T 集。返回 T 集的大小 \
，元素保存在传入的数组中。
*/
#include <cstdio>
#include <cstring>
#include <climits>
using namespace std;

const int MAXN=22000,MAXM=440000;
struct Dinic {
    struct edge {
        int x,y; //两个顶点
        int c; //容量
        int f; //当前流量
        edge *next,*back; //下一条边，反向边
        edge(int x,int y,int c,edge *next):x(x),y(y),c(c),f(0),next(next),back(0) {}
        void *operator new(size_t, void *p) {
            return p;
        }
    } *E[MAXN],*data; //E[i] 保存顶点 i 的边表
    char storage[2*MAXM *sizeof(edge)];
    int S,T; //源、汇

    int Q[MAXN]; //DFS 用到的 queue
    int D[MAXN]; //距离标号，-1 表示不可达
    void DFS() {
```

```
        memset(D,-1,sizeof(D));
        int head=0,tail=0;
        Q[tail++]=S;
        D[S]=0;
        for(;;) {
            int i=Q[head++];
            for(edge *e=E[i]; e; e=e->next) {
                if(e->c==0)continue;
                int j=e->y;
                if(D[j]==-1) {
                    D[j]=D[i]+1;
                    Q[tail++]=j;
                    if(j==T)return;
                }
            }
            if(head==tail)break;
        }
    }
    edge *cur[MAXN]; //当前弧
    edge *path[MAXN]; //当前找到的增广路
    int flow() {
        int res=0; //结果，即总流量
        int path_n; //path 的大小
        for(;;) {
            DFS();
            if(D[T]==-1)break;
            memcpy(cur,E,sizeof(E));
            path_n=0;
            int i=S;
            for(;;) {
                if(i==T) { //已找到一条增广路，增广之
                    int mink=0;
                    int delta=INT_MAX;
                    for(int k=0; k<path_n; ++k) {
                        if(path[k]->c < delta) {
                            delta = path[k]->c;
                            mink=k;
                        }
                    }
                    for(int k=0; k<path_n; ++k) {
                        path[k]->c -= delta;
                        path[k]->back->c += delta;
                    }
                    path_n=mink; //回退
                    i=path[path_n]->x;
                    res+=delta;
                }
                edge *e;
                for(e=cur[i]; e; e=e->next) {
                    if(e->c==0)continue;
                    int j=e->y;
                    if(D[i]+1==D[j])break; //找到一条弧，加到路径里
                }
```

```
                cur[i]=e; //当前弧结构, 访问过的不能增广的弧不会再访问
                if(e) {
                    path[path_n++]=e;
                    i=e->y;
                } else { //该节点已没有任何可增广的弧, 从图中删去, 回退一步
                    D[i]=-1;
                    if(path_n==0)break;
                    path_n--;
                    i=path[path_n]->x;
                }
            }
        }
        return res;
    }
    int cut(int *s) {
        int rst=0;
        for(int i=0; i<MAXN; ++i)
            if(D[i]==-1&&E[i])
                s[rst++]=i;
        return rst;
    }
    void init(int _S,int _T) {
        S=_S,T=_T;
        data=(edge *)storage;
        memset(E,0,sizeof(E));
    }
    void add_edge(int x,int y,int w) { //加进一条 x 至 y 容量为 w 的边, 需要保证 0<=x,y<MAXN, 0\
<w<=INT_MAX
        E[x]=new((void *)data++) edge(x,y,w,E[x]);
        E[y]=new((void *)data++) edge(y,x,0,E[y]);
        E[x]->back = E[y];
        E[y]->back = E[x];
    }
};


/**** 用来 AC POJ3469 的示范用法 ****/
Dinic dinic;
int main() {
    int N,M;
    while(2==scanf("%d%d",&N,&M)) {
        int rst=0;
        int S=0,T=N+1;
        dinic.init(S,T);
        for(int i=1; i<=N; ++i) {
            int a,b;
            scanf("%d%d",&a,&b);
            dinic.add_edge(S,i,a);
            dinic.add_edge(i,T,b);
        }
        for(int i=0; i<M; ++i) {
            int x,y,w;
            scanf("%d%d%d",&x,&y,&w);
            dinic.add_edge(x,y,w);
            dinic.add_edge(y,x,w);
```

```
        }
        rst=dinic.flow();
        printf("%d\n",rst);
    }
}
```

### 7.5.2 最高标号先流推进

```
//邻接表形式，邻接阵接口，复杂度 O(V^3)
//返回最大流量,f 返回每条边的流量，返回最大流量
//传入网络节点数 n，容量 mat，源点 s，汇点 t
const int MAXN = 210;
const int INF = 1000000000;

int maxFlow(int n, const int mat[][MAXN], int s, int t, int f[][MAXN]) {
    int g[MAXN][MAXN], cur[MAXN], h[MAXN], e[MAXN], q[MAXN], l[MAXN * 2][MAXN], i, j, k,\
 head, tail, ret, checked, p, o;
    for (i = 0; i < n; i++) {
        h[i] = - 1;
        cur[i] = 1;
        for (e[i] = g[i][0] = j = 0; j < n; f[i][j++] = 0) {
            if (mat[i][j] || mat[j][i]) {
                g[i][++g[i][0]] = j;
            }
        }
    }
    for (i = 0; i < 2 * n; i++) {
        l[i][0] = 0;
    }
    for (l[h[q[head = 0] = t] = 0][++l[0][0]] = t, tail = 1; head < tail; head++) {
        for (i = 1; i <= g[j = q[head]][0]; i++) {
            if (h[k = g[j][i]] < 0) {
                h[k] = h[j] + 1;
                q[tail++] = k;
                if (k != s) {
                    l[h[k]][++l[h[k]][0]] = k;
                }
            }
        }
    }
    for (h[s] = n, i = 1; i <= g[s][0]; i++) {
        j = g[s][i];
        f[j][s] = -(f[s][j] = e[j] = mat[s][j]);
    }
    for (i = n; i; i--) {
        for (checked = 0, j = l[i][0]; j;) {
            if ((k = l[i][j]) == s) {
                j--;
            } else if (!e[k]) {      //Full
                if (checked) {       //Update
                    if (i && l[i][0] == 1) {
                        for (p = h[k] + 1; p < n; l[n][0] += l[p][0], l[p++][0] = 0) {
                            for (o = 1; o <= l[p][0]; o++) {
                                l[n][++l[n][0]] = l[p][o];
```

```
                                h[l[p][o]] = n;
                            }
                        }
                    }
                    l[h[k]][++l[h[k]][0]] = k;
                    l[i][j] = l[i][l[i][0]--];
                    i = h[k];
                    break;
                } else {
                    j--;
                }
            } else if (cur[k] > g[k][0]) {  //Relabel
                for (checked = p = 1, o = INF; p <= g[k][0]; p++) {
                    if (mat[k][g[k][p]] > f[k][g[k][p]] && h[g[k][p]] < o) {
                        o = h[g[k][p]];
                    }
                }
                h[k] = o + 1, cur[k] = 1;
            } else if ((o = mat[k][p = g[k][cur[k]]] - f[k][p]) && h[k] == h[p] + 1) {
                o = o < e[k] ? o : e[k];
                f[p][k] = -(f[k][p] += o);
                e[k] -= o;
                e[p] += o;
            } else {
                cur[k]++;
            }
        }
    }
    for (ret = 0, i = 1; i <= g[s][0]; i++) {
        ret += f[s][g[s][i]];
    }
    return ret;
}
```

### 7.5.3 网络流 (全功能)

```
// 通用网络流 (Sap + Johnson + 上下界) By 猛犸也钻地 @ 2012.02.10

#include <cstring>
#include <queue>
#include <algorithm>
using namespace std;

class Network {
public:
    typedef int VAL;    // 费用的类型
    static const int SIZE = 1005;       // 最大点数
    static const int INF = 1000000007;  // 流量的极大值
    typedef struct ARC {
        int t,c;
        VAL w;
        ARC *o;
    } *PTR;
    ARC arc[200005];    // 最大边数，注意一次普通加边操作需要占用两条边
```

```
    PTR now[SIZE],e[SIZE];      // cnt[] 为该层次下的点数, l[] 为层次标号
    int cnt[SIZE],l[SIZE],r[SIZE],edge; // now[] 为当前弧, e[] 为出边链表
    VAL sum;  // sum 为当前流网络下的费用
    void clear() {
        memset(e,edge=sum=0,sizeof(e));   // 清空边表
    }
    ARC &REV(PTR x) {
        return arc[(x-arc)^1];   // 取反向边
    }
    // 传入源点 S 和汇点 T, 返回流量, 处理费用流时把下面改成 spfa_johnson
    int flow(int S, int T) {
        return improved_sap(S,T,INF);
    }
    // 加入一条 x 到 y 的有向边, 容量为 c, 费用为 w
    PTR add_edge(int x, int y, int c, VAL w = 0) {
        e[x]=&(arc[edge++]=(ARC) {
            y,c,+w,e[x]
        });
        e[y]=&(arc[edge++]=(ARC) {
            x,0,-w,e[y]
        });
        return e[x];
    }
    // 加入一条 x 到 y 的无向边, 容量为 c, 费用为 0
    PTR add_edge_simple(int x, int y, int c) {
        e[x]=&(arc[edge++]=(ARC) {
            y,c,0,e[x]
        });
        e[y]=&(arc[edge++]=(ARC) {
            x,c,0,e[y]
        });
        return e[x];
    }
    // 加入一条 x 到 y 的有向边, 下界为 lo, 上界为 hi, 费用为 w
    // 超级源在 SIZE-2, 超级汇在 SIZE-1, 注意给这两个点预留空间
    PTR add_edge_bounded(int x, int y, int lo, int hi, VAL w = 0) {
        add_edge(SIZE-2,y,lo,w);
        add_edge(x,SIZE-1,lo,0);
        return add_edge(x,y,hi-lo,w);
    }
    // 对 S 至 T 且出弧为 now[] 的增广路进行松弛, 返回被阻塞的结点
    int aug(int S, int T, int &can) {
        int x,z=T,use=can;
        for(x=S; x!=T; x=now[x]->t) if(use>now[x]->c) use=now[z=x]->c;
        for(x=S; x!=T; x=now[x]->t) {
            now[x]->c-=use;
            REV(now[x]).c+=use;
            sum+=use*now[x]->w;
        }
        can-=use;
        return z;
```

```cpp
    }
    // 无权值最短路增广算法，用在无费用的网络流上，返回流量
    int improved_sap(int S, int T, int can) { // can 为本次增广的流量上限
        if(S==T) return can;
        int in=can,x,m;
        memcpy(now,e,sizeof(now));
        memset(cnt,0,sizeof(cnt));
        fill_n(l,SIZE,int(SIZE));
        for(int i=m=l[r[0]=T]=0; i<=m; i++) {
            cnt[l[x=r[i]]]++;
            for(PTR u=e[x]; u; u=u->o)
                if(l[u->t]==SIZE && REV(u).c) l[r[++m]=u->t]=l[x]+1;
        }
        for(x=r[S]=S; l[S]!=SIZE; x=r[x]) {
JMP:
            for(PTR &u=now[x]; u; u=u->o) if(l[u->t]<l[x] && u->c) {
                    r[u->t]=x;
                    x=u->t==T?aug(S,T,can):u->t;
                    if(x==T) return in;
                    else goto JMP;
                }
            if(!--cnt[l[x]]) break;
            else l[x]=SIZE;
            for(PTR u=e[x]; u; u=u->o)
                if(l[u->t]+1<l[x] && u->c) now[x]=u,l[x]=l[u->t]+1;
            if(l[x]!=SIZE) cnt[l[x]]++;
        }
        return in-can;
    }
    // 连续最短路增广算法，可以处理不含负费用圈的费用流，返回流量
    int spfa_johnson(int S, int T, int can) { // can 为本次增广的流量上限
        if(S==T) return can;
        int in=can,x,m;
        VAL phi[SIZE],len[SIZE],MAXW=1000000007; // 费用的极大值
        memset(l,0,sizeof(l));
        fill_n(phi,SIZE,MAXW);
        phi[r[0]=T]=0;
        for(int i=m=0; i<=m; i++) { // 从汇点出发反向 SPFA
            l[x=r[i%SIZE]]=0;
            for(PTR u=e[x]; u; u=u->o) { // 下面这行如果是浮点比较要加 EPS
                if(phi[x]+REV(u).w>=phi[u->t] || !REV(u).c) continue;
                phi[u->t]=phi[x]+REV(u).w;
                if(!l[u->t]) l[r[++m%SIZE]=u->t]=1;
            }
        }
        do {
            typedef pair<VAL,int> TPL;
            priority_queue<TPL> q;
            fill_n(len,SIZE,MAXW);
            memset(l,0,sizeof(l));
            q.push(TPL(len[T]=0,T));
            while(q.size()) {
                x=q.top().second;
```

```
                q.pop();
                if(!l[x]) l[x]=1;
                else continue;
                for(PTR u=e[x]; u; u=u->o) {
                    if(!REV(u).c || l[u->t]) continue;
                    VAL at=len[x]+phi[x]+REV(u).w-phi[u->t];
                    if(at>=len[u->t]) continue; // 如果是浮点比较要加 EPS
                    len[u->t]=at;
                    now[u->t]=&REV(u);
                    q.push(TPL(-at,u->t));
                }
            }
            for(x=0; x<SIZE; x++) phi[x]+=len[x];
        } while(phi[S]<MAXW && aug(S,T,can)!=T);
        // 使用 phi[S]<MAXW 求最小费用最大流，使用 phi[S]<0 求最小费用流
        return in-can;
    }
    // 判断无源汇上下界可行流是否存在
    // 加入边 (T,S)=MAXF 可处理带源汇的情况，此时反向弧 S->T 的 c 即为流量
    bool feasible_bounded() {
        flow(SIZE-2,SIZE-1);
        for(PTR u=e[SIZE-2]; u; u=u->o) if(u->c) return false;
        return true;
    }
    // 有源汇上下界最大/最小流，返回 -1 表示不存在可行流，否则返回流量
    int max_flow_bounded(int S, int T) {
        add_edge(T,S,INF);
        bool ok=feasible_bounded();
        int ret=e[S]->c;
        edge-=2,e[S]=e[S]->o,e[T]=e[T]->o;
        return ok?ret+flow(S,T):-1;
    }
    int min_flow_bounded(int S, int T) {
        flow(SIZE-2,SIZE-1);
        add_edge(T,S,INF);
        bool ok=feasible_bounded();
        int ret=e[S]->c;
        edge-=2,e[S]=e[S]->o,e[T]=e[T]->o;
        return ok?ret:-1;
    }
    // 将所有带下界的边合并回原图中
    void merge_bounded() {
        for(PTR u=e[SIZE-1]; u; u=u->o) u->c=REV(u).c=0;
        for(PTR u=e[SIZE-2]; u; u=u->o) {
            (u+4)->c+=u->c;
            (u+5)->c+=REV(u).c;
            u->c=REV(u).c=0;
        }
    }
};
```

# 7.6 连通性

## 7.6.1 支配树

```
/* 最近必经点 (idom): 节点 y 的必经点集合 dom(y) 中 dfn 值最大的点 x 是距离 y 最近的必经点
                          称为 y 的最近必经点, 最近必经点是唯一的, 记 x=idom(y)
   半必经点 (semi): 在搜索树 T 上点 y 的祖先中, 通过非树枝边可以到达 y 的深度最小的祖先 x,
                        称为 y 的半必经点。半必经点也是唯一的, 记 x=semi(y)。

   1. 设有向图 G=(V,E), (G,r) 是一个 Flow Graph, 则称 (G,r) 的子图
      D=(V, { (idom(i),i) | i∈V,i≠r }, r) 为 (G,r) 的一棵 Dominator Tree。
   2. (G,r) 的 Dominator Tree 是一棵有向有根树, 从 r 出发可以到达 G 中的所有点,
      并且树上的每条边 (u,v) 都满足: u=idom(v), 即父节点是子节点的最近必经点。
   3. x=idom(y), 当且仅当有向边 (x,y) 是 Dominator Tree 中的一条树枝边。
   4. x dom y, 当且仅当在 Dominator Tree 中存在一条从 x 到 y 的路径。
   5. x 的必经点集合 dom(x) 就是 Dominator Tree 上 x 的所有祖先以及 x 自身。
*/

/* 应用:
1. 求有向图的割顶: dominator tree 上的非叶节点
2. 有向图的必经边: 每条边上加一个点, 转化成必经点问题
3. 求起点 S 到终点 T 的所有路径中最接近源的必经点: 求出必经点, 取最近的
4. 求多少个 (x,y) 满足存在 1->x 的路径和 1->y 的路径只有 1 这个公共点:
      求出 1 为根的 dominator tree, 算出不合法的, 总的减去即可.
      考虑 1 的每个儿子 v, 同一颗子树的节点对都是非法的.
*/
//succ 是原图, pred 是边反向后的图, dom 是 Dominator Tree
//dom 记录的是 dfs 序构成的树, G 中节点 u 在 dom 树上的标号是 dfn[u]
//相反 dom 中节点 u 在原图 G 中的标号是 pt[u]
//调用 build 得到以 s 为根的 Dominator Tree
namespace DominatorTree {
int dfn[MAXN], pre[MAXN], pt[MAXN], sz;
int semi[MAXN], dsu[MAXN], idom[MAXN], best[MAXN];
int get(int x) {
    if (x == dsu[x]) return x;
    int y = get(dsu[x]);
    if (semi[best[x]] > semi[best[dsu[x]]]) best[x] = best[dsu[x]];
    return dsu[x] = y;
}
void dfs(int u, const VI succ[]) {
    dfn[u] = sz;
    pt[sz ++] = u;
    for (auto &v: succ[u]) if (!~dfn[v]) {
```

```
            dfs(v, succ);
            pre[dfn[v]] = dfn[u];
        }
}
void tarjan(const VI pred[], VI dom[]) {
    for (int j = sz - 1, u; u = pt[j], j > 0; -- j) {
        for (auto &tv: pred[u]) if (~dfn[tv]) {
                int v = dfn[tv];
                get(v);
                if (semi[best[v]] < semi[j]) semi[j] = semi[best[v]];
            }
        dom[semi[j]].push_back(j);
        int x = dsu[j] = pre[j];
        for (auto &z: dom[x]) {
            get(z);
            if (semi[best[z]] < x) idom[z] = best[z];
            else idom[z] = x;
        }
        dom[x].clear();
    }
    for (int i = 1; i < sz; ++ i) {
        if (semi[i] != idom[i]) idom[i] = idom[idom[i]];
        dom[idom[i]].push_back(i);
    }
}
void build(int n, int s, const VI succ[], const VI pred[], VI dom[]) {
    for (int i = 0; i < n; ++ i) {
        dfn[i] = -1;
        dom[i].clear();
        dsu[i] = best[i] = semi[i] = i;
    }
    sz = 0;
    dfs(s, succ);
    tarjan(pred, dom);
}
}
```

### 7.6.2 无向图关键点、关键边和块

```
//无向图关键点、关键边和双连通块（由关键点分割得到的块）
//复杂度 O(m)，边存在 E 中，要保证传入时的 E 没有重边
//调用 ArtEdge_ArtVertex_Components(总点数)
//结果关键点储存在 keyV 中
//结果关键边储存在 keyE 中
#include <cstring>
#include <vector>
using namespace std;
#define MP(i,j) make_pair(i, j)
#define MAXN 10000
int dfn[MAXN], low[MAXN], st[MAXN], top;
vector <pair<int, int> > keyE;
vector <int> keyV, E[MAXN];

void tarjan(int now, int cnt) {
```

```cpp
    int part = (cnt > 1);
    st[top++] = now;
    dfn[now] = low[now] = cnt;
    for (int ii = E[now].size() - 1; ii >= 0; --ii) {
        int i = E[now][ii];
        if (!dfn[i]) {
            tarjan(i, cnt + 1);
            low[now] = min(low[now], low[i]);
            if (low[i] > dfn[now])          // 这两行用于求取关键边
                keyE.push_back(MP(now, i));

            if (low[i] >= dfn[now]) {
                if(++part == 2)             // 以下两行求取关键点
                    keyV.push_back(now);

                vector <int> A;             //以下四行求取双联通块
                A.push_back(now);
                for (st[top] = 0; st[top] != i; A.push_back(st[--top]));
                dummy(A); //每次 dummy 调用的 A 中包含一个联通块
            }
        } else if (dfn[i] != dfn[now] - 1)
            low[now] = min(low[now], dfn[i]);
    }
    //此时 part 值等于将此点去掉之后它原先所在的联通块分成的块数
}
void ArtEdge_ArtVertex_Components(int N) {
    memset(dfn, 0, sizeof(dfn));
    memset(low, 0, sizeof(low));
    keyE.clear();
    keyV.clear();
    for (int i = top = 0; i < N; ++i)
        if (!dfn[i])
            tarjan(i, 1);
}
```

### 7.6.3 有向图强连通分量

```cpp
//有向图强连通分量，复杂度 O(m)
//边存在 E 中，调用 Components(总点数)
//返回 id 数组储存每个点所在的强连通分量编号 (从 0 开始)
//最后 num 表示一共有几个强连通分量
#include <cstring>
#include <vector>
using namespace std;
#define MP(i,j) make_pair(i, j)
#define MAXN 10000
int dfn[MAXN], low[MAXN], id[MAXN], num, st[MAXN], top, in[MAXN], tot;
vector <int> E[MAXN];

void tarjan(int now) {
    in[st[top++] = now] = true;
    dfn[now] = low[now] = ++tot;
```

```
    int i;
    for (int ii = E[now].size() - 1; ii >= 0; --ii) {
        i = E[now][ii];
        if (!dfn[i]) {
            tarjan(i);
            low[now] = min(low[now], low[i]);
        } else if (in[i])
            low[now] = min(low[now], dfn[i]);
    }
    if (dfn[now] == low[now]) {
        do {
            i = st[--top];
            in[i] = false;
            id[i] = num;
        } while (i != now);
        ++num;
    }
}
void Components(int N) {
    memset(dfn, 0, sizeof(dfn));
    memset(low, 0, sizeof(low));
    memset(in, 0, sizeof(in));
    memset(id, 0xff, sizeof(id));
    for (int i = top = num = tot = 0; i < N; ++i)
        if (!dfn[i])
            tarjan(i);
}
```

# Chapter 8

# 应用

## 8.1 简单位操作

| 功能 | 示例 | 位运算 | 备注 |
|---|---|---|---|
| 把右数第 k 位置 0 | | x & (~(1 << k)) | k 从 0 开始计数 |
| 把右数第 k 位置 1 | | x \| (1 << k) | k 从 0 开始计数 |
| 把右数第 k 位取反 | | x ^ (1 << k) | k 从 0 开始计数 |
| 得到右数第 k 位值 | | (x >> k) & 1 | k 从 0 开始计数 |
| 得到末尾 k 位 | | x & ((1 << k) - 1) | |
| 把最右边的 1 置 0 | 01011000 -> 01010000 | x & (x - 1) | 把结果跟 0 作比较可以得出 x 是否为 2 的幂 |
| 得到最右边的 1 的掩码 | 01011000 -> 00001000 | x & (-x) | |
| 把最右边的 0 置 1 | 01011000 -> 01011001 | x \| (x + 1) | |
| 得到最右边的 0 的掩码 | 10100111 -> 00001000 | (~x) & (x + 1) | |
| 把末尾连续 0 串置 1 | 01011000 -> 01011111 | x \| (x - 1) | 如果 x 为 0 则结果为全 1 |

Continued on next page

<table>
<tr><th>功能</th><th>示例</th><th>位运算</th><th>备注</th></tr>
<tr><td>得到末尾连续 0 串的掩码</td><td>01011000 -> 00000111</td><td>(~x) & (x - 1)</td><td>或者使用 ~(x | (-x)) 和 (x & (-x)) - 1</td></tr>
<tr><td>得到最右边 1 及其右边连续 0 串的掩码</td><td>01011000 -> 00001111</td><td>x ^ (x - 1)</td><td>如果 x 为 0 则结果为全 1</td></tr>
<tr><td>把最右边的连续 1 串置 0</td><td>01011000 -> 01000000</td><td>((x | (x - 1)) + 1) & x</td><td>把结果跟 0 作比较可以得出 x 是否为 $2^j - 2^k$ (j ≥ k ≥ 0)</td></tr>
<tr><td>得到最右边的连续 1 串的掩码</td><td>01011000 -> 00011000</td><td>(((x | (x - 1)) + 1) & x) ^ x</td><td></td></tr>
<tr><td>下舍入到 2k 倍数</td><td></td><td>x & ((-1) << k)</td><td></td></tr>
<tr><td>上舍入到 2k 倍数</td><td></td><td>t = (1 << k) - 1; x = (x + t) & (~t)</td><td></td></tr>
</table>

GCC 内建函数, 接受 unsigned int, 返回 int:

- `__builtin_ffs`
  二进制末尾 (最低位开始) 第一个 1 的 index (从 1 开始), x 是 0 时返回 0
- `__builtin_clz`
  二进制开头 (最高位开始) 连续的 0 的个数, x 不能是 0
- `__builtin_ctz`
  二进制末尾 (最低位开始) 连续的 0 的个数, x 不能是 0
- `__builtin_popcount`
  二进制 1 的个数
- `__builtin_parity`
  奇偶校验, `__builtin_popcount` % 2

## 8.2 Joseph

```
// 编号 1 to n
// 每次第 d 个人出局
// 返回剩下的人的编号
int joseph(int n, int d) {
    int ret = 1;
    for (int i = 2; i <= n; i++)
        ret = (ret + d - 1) % i + 1;
    return ret;
}

//Joseph 问题模拟
//传入 n,m, r 返回出环的序列
//时间复杂度 O(nlogn)
#include <cstring>

const int MAXN = 32768;

void josephus(int n, int m, int r[]) {
    int d[MAXN * 2], i, j, nbase, p, t;
    for (nbase = 1; nbase < n; nbase <<= 1);
    memset(d, 0, sizeof(d));
    for (i = 0; i < n; i++) {
        d[nbase + i] = 1;
    }
```

```
    for (i = nbase - 1; i; i--) {
        d[i] = d[i << 1] + d[i << 1 | 1];
    }
    for (j = i = 0; i < n; i++) {
        for (j = t = (j - 1+m) % (n - i), p = 1; p < nbase;) {
            d[p]--;
            if (t < d[p << 1]) {
                p = p << 1;
            } else {
                t -= d[p << 1];
                p = p << 1 | 1;
            }
        }
        r[i] = p - nbase;
        d[p]--;
    }
}
```

## 8.3 位操作

```
// 遍历一个掩码的所有子集掩码，不包括 0 和其自身
// 传入表示超集的掩码
void iterateSubset(int mask) {
    for(int sub = (mask - 1) & mask; sub > 0; sub = (sub - 1) & mask) {
        int incsub = ~sub & mask; // 递增顺序的子集
        // gogogo
    }
}

// 下一个包含同样数量二进制 1 的掩码
// 传入掩码，掩码不能为 0
unsigned snoob(unsigned mask) {
    unsigned smallest, ripple, ones;
    // x = xxx0 1111 0000
    smallest = mask & -mask;          //     0000 0001 0000
    ripple = mask + smallest;         //     xxx1 0000 0000
    ones = mask ^ ripple;             //     0001 1111 0000
    ones = (ones >> 2) / smallest;    //     0000 0000 0111
    return ripple | ones;             //     xxx1 0000 0111
}

// 遍历 {0, 1, ⋯ , n-1} 的所有 k 元子集
// 传入 n 和 k,k 不为 0
void iterateSubset(int n, int k) {
    int s = (1 << k) - 1;
    while (!(s & 1 << n)) {
        // gogogo
```

```
        s = snoob(s);
    }
}

// 求一个 32 位整数二进制 1 的位数
// 初始化函数先调用一次
int ones[256];

void initOnes() {
    for (int i = 1; i < 256; ++i)
        ones[i] = ones[i & (i - 1)] + 1;
}

int countOnes(int n) {
    return ones[n & 255] + ones[(n >> 8) & 255] + ones[(n >> 16) & 255] + ones[(n >> 24)\
 & 255];
}

// 求 32 位整数二进制 1 的位数，不需要初始化 (from Beautiful Code 2007 Ch. 10)
int countOnes(unsigned x) {
    x = x - ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x + (x >> 4)) & 0x0F0F0F0F;
    x = x + (x >> 8);
    x = x + (x >> 16);
    return x & 0x0000003F;
}


// 求一个 32 位整数二进制 1 的位数的奇偶性
// 偶数返回 0，奇数返回 1
int parityOnes(unsigned n) {
    n ^= n >> 1;
    n ^= n >> 2;
    n ^= n >> 4;
    n ^= n >> 8;
    n ^= n >> 16;
    return n & 1; // n 的第 i 位是原数第 i 位到最左侧位的奇偶性
}

// 对一个 32 位整数进行位反转
unsigned revBits(unsigned n) {
    n = (n & 0x55555555) << 1 | (n >> 1) & 0x55555555;
    n = (n & 0x33333333) << 2 | (n >> 2) & 0x33333333;
    n = (n & 0x0F0F0F0F) << 4 | (n >> 4) & 0x0F0F0F0F;
    n = (n << 24) | ((n & 0xFF00) << 8) | ((n >> 8) & 0xFF00) | (n >> 24);
    return n;
}

// 对 n 个数根据二进制掩码求部分和，类似的可以用于求 gcd，求二进制 1 的个数等等
void partSum(int n, int a[], int s[]) {
    s[0] = 0;
    for (int i = 0; i < n; i++) {
```

```
        s[1 << i] = a[i];
    }
    for (int i = 1; i < (1 << n); i++) {
        s[i] = s[i&(i-1)] + s[i^(i&(i-1))];
    }
}
```

## 8.4 布尔母函数

```
//布尔母函数
//判 m[] 个价值为 w[] 的货币能否构成 value
//适合 m[] 较大 w[] 较小的情况
//返回布尔量
//传入货币种数 n, 个数 m[], 价值 w[] 和目标值 value
const int MAXV = 100000;

bool genfunc(int n, const int *m, const int *w, int value) {
    int i, j, k, c;
    char r[MAXV];
    for (r[0] = i = 1; i <= value; r[i++] = 0);
    for (i = 0; i < n; i++) {
        for (j = 0; j < w[i]; j++) {
            c = m[i] * r[k = j];
            while ((k += w[i]) <= value) {
                if (r[k]) {
                    c = m[i];
                } else if (c) {
                    r[k] = 1;
                    c--;
                }
            }
            if (r[value]) {
                return true;
            }
        }
    }
    return false;
}
```

## 8.5 快速沃尔什哈达马变换

```
/*
 * FWT, 默认为 xor 运算, 要修改为 and 或 or 的时候按注释改即可。
 */
#include<bits/stdc++.h>
using namespace std;
#define rep(i,s,t) for (int i=s;i<=t;i++)
#define FOR(i,s,t) for (int i=s;i<t;i++)
const int maxn = 1<<12;
const LL mod = 1e9 + 7;
LL a[maxn], b[maxn];
```

```
LL P = mod;
LL PowMod(LL x, LL k) {
    LL ret = 1;
    for (; k; k >>= 1, x = x * x % mod)
        if (k & 1) ret = (ret * x) % mod;
    return ret;
}
LL inv2 = PowMod(2, mod - 2);
void TransForm(int l, int r, LL a[]) { /// [l, r)
    if (l == r - 1) return;
    int len = (r - l) >> 1;
    int m = l + len;
    TransForm(l, m, a);
    TransForm(m, r, a);
    FOR(i,l,m) {
        LL a0 = a[i];
        LL a1 = a[i+len];
        a[i] = (a0 - a1 + mod) % mod;      /* And: a[i] = a0 + a1;    Or: a[i] = a0;*/
        a[i+len] = (a0 + a1) % mod;        /* a[i+len] = a1;          a[i+len] = a0 + a\
1; */
    }
}
void uTransForm(int l, int r, LL a[]) {
    if (l == r - 1) return;
    int len = (r - l) >> 1;
    int m = l + len;
    FOR(i,l,m) {
        LL y0 = a[i];
        LL y1 = a[i + len];
        a[i] = (y0 + y1) * inv2 % mod;     /* And: a[i] = y0 - y1     Or: a[i] = y0 */
        a[i+len] = (y1 - y0 + mod) * inv2 % mod; /* a[i+len] = y1     a[i+len] = y1 - y\
0 */
    }
    uTransForm(l, m, a);
    uTransForm(m, r, a);
}
void Conv(LL a[], LL b[], int len) { // 请保证 len 为 2 的幂次
    TransForm(0, len, a);
    TransForm(0, len, b);
    FOR(i,0,len) a[i] = a[i] * b[i] % mod;
    uTransForm(0, len, a);
}
int main() { // 用法如下
    rep(i,0,4) a[i] = i + 1;
    rep(i,0,4) b[i] = 4 - i;
    Conv(a, b, 1 << 11);
    rep(i,0,8) cout << i << " " << a[i] << endl;
}
```

## 8.6 快速线性递推

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

typedef long long LL;

const int M = 2;

// written by yxdb
// given first m a[i] and coef c[i] (0-based),
// calc a[n] mod p in O(m*m*log(n)).
// a[n] = sum(c[m-i]*a[n-i]), i = 1...m
// i.e. a[m] = sum(c[i]*a[i]), i = 0...m-1
int linear_recurrence(LL n, int m, int a[], int c[], int p) {
    LL v[M] = {1 % p}, u[M<<1], msk = !!n;
    for(LL i = n; i > 1; i >>= 1) msk <<= 1;
    for(LL x = 0; msk; msk >>= 1, x <<= 1) {
        fill_n(u, m<<1, 0);
        int b = !!(n & msk);
        x |= b;
        if(x < m) u[x] = 1 % p;
        else {
            for(int i = 0; i < m; ++i)
                for(int j = 0, t = i+b; j < m; ++j, ++t)
                    u[t] = (u[t]+v[i]*v[j]) % p;
            for(int i = (m<<1)-1; i >= m; --i)
                for(int j = 0, t = i-m; j < m; ++j, ++t)
                    u[t] = (u[t]+c[j]*u[i]) % p;
        }
        copy(u, u+m, v);
    }
    int an = 0;
    for(int i = 0; i < m; ++i) an = (an+v[i]*a[i]) % p;
    return an;
}

const int MOD = 1000000007;

int main() {
    int n, a[2] = {0, 1}, c[2] = {1, 1};
    while(1==scanf("%d", &n) && n >= 0) {
        printf("%d\n", linear_recurrence(n, 2, a, c, MOD));
    }
}
```

## 8.7 最大子阵和

```
//求最大子阵和, 复杂度 O(n^3)
//传入阵的大小 m,n 和内容 mat[][]
```

```
//返回最大子阵和，重载返回子阵位置 (maxsum=list[s1][s2]+...+list[e1][e2])
//可更改元素类型
const int MAXN = 100;

template <class elemType>
elemType maxsum(int m, int n, const elemType mat[][MAXN]) {
    elemType matsum[MAXN][MAXN + 1], ret, sum;
    int i, j, k;
    for (i = 0; i < m; i++) {
        for (matsum[i][j = 0] = 0; j < n; j++) {
            matsum[i][j + 1] = matsum[i][j] + mat[i][j];
        }
    }
    for (ret = mat[0][j = 0]; j < n; j++) {
        for (k = j; k < n; k++) {
            for (sum = 0, i = 0; i < m; i++) {
                sum = (sum > 0 ? sum : 0) + matsum[i][k + 1] - matsum[i][j], ret = (sum \
> ret ? sum : ret);
            }
        }
    }
    return ret;
}

template <class elemType>
elemType maxsum(int m, int n, const elemType mat[][MAXN], int &s1, int &s2, int &e1, int\
 &e2) {
    elemType matsum[MAXN][MAXN + 1], ret, sum;
    int i, j, k, s;
    for (i = 0; i < m; i++) {
        for (matsum[i][j = 0] = 0; j < n; j++) {
            matsum[i][j + 1] = matsum[i][j] + mat[i][j];
        }
    }
    for (ret = mat[s1 = e1 = 0][s2 = e2 = j = 0]; j < n; j++) {
        for (k = j; k < n; k++) {
            for (sum = 0, s = i = 0; i < m; i++, s = (sum > 0 ? s : i)) {
                if ((sum = (sum > 0 ? sum : 0) + matsum[i][k + 1] - matsum[i][j]) > ret)\
 {
                    ret = sum;
                    s1 = s;
                    s2 = i;
                    e1 = j;
                    e2 = k;
                }
            }
        }
    }
    return ret;
}
```

## 8.8 最长公共单调子序列

```
/**
 * 最长公共递增子序列, 时间复杂度 O(n^2 * logn), 空间 O(n^2)
```

```
 * n 为 a 的大小, m 为 b 的大小
 * 结果在 ans 中
 * "define _cp(a,b) ((a)<(b))" 求解最长严格递增序列
 */
const int MAXN = 1000;
#define _cp(a,b) ((a) < (b))

typedef int elemType;

elemType DP[MAXN][MAXN];
int num[MAXN], p[1 << 20];
int lis(int n, const elemType *a, int m, const elemType *b, elemType *ans) {
    int i, j, l, r, k;
    DP[0][0] = 0;
    num[0] = (b[0] == a[0]);
    for (i = 1; i < m; i++) {
        num[i] = (b[i] == a[0]) || num[i - 1];
        DP[i][0] = 0;
    }
    for (i = 1; i < n; i++) {
        if (b[0] == a[i] && !num[0]) {
            num[0] = 1;
            DP[0][0] = i << 10;
        }
        for (j = 1; j < m; j++) {
            for (k = ((l = 0) + (r = num[j - 1] - 1)) >> 1; l <= r; k = (l + r) >> 1) {
                if (_cp(a[DP[j - 1][k] >> 10], a[i])) {
                    l = k + 1;
                } else {
                    r = k - 1;
                }
            }
            if (l < num[j - 1] && i == (DP[j - 1][l] >> 10)) {
                if (l >= num[j]) {
                    DP[j][num[j]++] = DP[j - 1][l];
                } else {
                    DP[j][l] = _cp(a[DP[j][l] >> 10], a[i]) ? DP[j][l] : DP[j - 1][l];
                }
            }
            if (b[j] == a[i]) {
                for (k = ((l = 0) + (r = num[j] - 1)) >> 1; l <= r; k = (l + r) >> 1) {
                    if (_cp(a[DP[j][k] >> 10], a[i])) {
                        l = k + 1;
                    } else {
                        r = k - 1;
                    }
                }
                DP[j][l] = (i << 10) + j;
                num[j] += (l >= num[j]);
                p[DP[j][l]] = l ? DP[j][l - 1]:  - 1;
            }
        }
    }

    for (k = DP[m - 1][i = num[m - 1] - 1]; i >= 0; k = p[k]) {
        ans[i--] = a[k >> 10];
```

```
    }
    return num[m - 1];
}
```

## 8.9 最长子序列

```
//最长单调子序列，复杂度 O(nlogn)
//注意最小序列覆盖和最长序列的对应关系，例如
//"define _cp(a,b) ((a)>(b))" 求解最长严格递减序列，则
//"define _cp(a,b) (!((a)>(b)))" 求解最小严格递减序列覆盖
//可更改元素类型和比较函数
const int MAXN = 10000;
#define _cp(a,b) ((a)>(b))

template <class elemType>
int subseq(int n, const elemType *a) {
    int b[MAXN + 1], i, l, r, m, ret = 0;
    for (i = 0; i < n; b[l] = i++, ret += (l> ret)) {
        for (m = ((l = 1) + (r = ret)) >> 1; l <= r; m = (l + r) >> 1) {
            if (_cp(a[b[m]], a[i])) {
                l = m + 1;
            } else {
                r = m - 1;
            }
        }
    }
    return ret;
}

template <class elemType>
int subseq(int n, const elemType *a, elemType *ans) {
    int b[MAXN + 1], p[MAXN], i, l, r, m, ret = 0;
    for (i = 0; i < n; p[b[l] = i++] = b[l - 1], ret += (l> ret)) {
        for (m = ((l = 1) + (r = ret)) >> 1; l <= r; m = (l + r) >> 1) {
            if (_cp(a[b[m]], a[i])) {
                l = m + 1;
            } else {
                r = m - 1;
            }
        }
    }
    for (m = b[i = ret]; i; ans[--i] = a[m], m = p[m]);
    return ret;
}
```

## 8.10 行列式求模

```
// @author Navi
// 高斯消元法行列式求模。复杂度 O(n^3logn)。
// n 为行列式大小，计算 |mat| % m
const int MAXN = 200;
```

```cpp
typedef long long LL;
int detMod(int n, int m, int mat[][MAXN]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            mat[i][j] %= m;
    LL ret = 1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++)
            while (mat[j][i] != 0) {
                LL t = mat[i][i] / mat[j][i];
                for (int k = i; k < n; k++) {
                    mat[i][k] = (mat[i][k] - mat[j][k] * t) % m;
                    int s = mat[i][k];
                    mat[i][k] = mat[j][k];
                    mat[j][k] = s;
                }
                ret = -ret;
            }
        if (mat[i][i] == 0)
            return 0;
        ret = ret * mat[i][i] % m;
    }
    if (ret < 0)
        ret += m;
    return (int)ret;
}

// @author Navi
// 高斯消元法行列式求模。复杂度 O(n^3 + n^2logn)。
// n 为行列式大小，计算 |mat| % m
// 速度只比 O(n^3logn) 的快一些，推荐用另外那个。
const int MAXN = 200;
typedef long long LL;
int detMod(int n, int m, int mat[][MAXN]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            mat[i][j] %= m;
    LL ret = 1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            LL x1 = 1, y1 = 0, x2 = 0, y2 = 1, p = i, q = j;
            if (mat[i][i] < 0) {
                x1 = -1;
                mat[i][i] = -mat[i][i];
                ret = -ret;
            }
            if (mat[j][i] < 0) {
                y2 = -1;
                mat[j][i] = -mat[j][i];
                ret = -ret;
            }
            while (mat[i][i] != 0 && mat[j][i] != 0) {
                if (mat[i][i] <= mat[j][i]) {
                    int t = mat[j][i] / mat[i][i];
                    mat[j][i] -= mat[i][i] * t;
```

```
                    x2 -= x1 * t;
                    y2 -= y1 * t;
                } else {
                    int t = mat[i][i] / mat[j][i];
                    mat[i][i] -= mat[j][i] * t;
                    x1 -= x2 * t;
                    y1 -= y2 * t;
                }
            }
            x1 %= m;
            y1 %= m;
            x2 %= m;
            y2 %= m;
            if (mat[i][i] == 0 && mat[j][i] != 0) {
                ret = -ret;
                p = j;
                q = i;
                mat[i][i] = mat[j][i];
                mat[j][i] = 0;
            }
            for (int k = i + 1; k < n; k++) {
                int s = mat[i][k], t = mat[j][k];
                mat[p][k] = (s * x1 + t * y1) % m;
                mat[q][k] = (s * x2 + t * y2) % m;
            }
        }
        if (mat[i][i] == 0)
            return 0;
        ret = ret * mat[i][i] % m;
    }
    if (ret < 0)
        ret += m;
    return (int)ret;
}
```

## 8.11 逆序对数

```
//序列逆序对数，复杂度 O(nlogn)
//传入序列长度和内容，返回逆序对数
//可更改元素类型和比较函数
#include <cstring>
const int MAXN = 1000000;
#define _cp(a,b) ((a) <= (b))

typedef int elemType;
elemType tmp[MAXN];

long long inv(int n, elemType *a) {
    int left = n >> 1, r = n - left, i, j;
    long long ret = (r > 1 ? (inv(left, a) + inv(r, a + left)) : 0);
    for (i = j = 0; i <= left; tmp[i + j] = a[i], i++) {
        for (ret += j; j < r && (i == left || !_cp(a[i], a[left + j])); tmp[i + j] = a[l\
eft + j], j++);
    }
    memcpy(a, tmp, sizeof(elemType) * n);
```

```
    return ret;
}
```

# Chapter 9

# 其他

## 9.1 分数

```
#include <cmath>

struct Frac {
    int num, den;
};

int gcd(int a, int b) {
    int t;
    if (a < 0) {
        a = -a;
    }
    if (b < 0) {
        b = -b;
    }
    if (!b) {
        return a;
    }
    while (t = a % b) {
        a = b;
        b = t;
    }
    return b;
}

void simplify(Frac &f) {
    int t;
    if (t = gcd(f.num, f.den)) {
        f.num /= t;
        f.den /= t;
    } else {
        f.den = 1;
    }
}

Frac f(int n, int d, int s = 1) {
    Frac ret;
    if (d < 0) {
        ret.num = -n;
```

```
        ret.den = -d;
    } else {
        ret.num = n;
        ret.den = d;
    }
    if (s) {
        simplify(ret);
    }
    return ret;
}

Frac convert(double x) {
    Frac ret;
    for (ret.den = 1; fabs(x - int(x)) > 1e-10; ret.den *= 10, x *= 10);
    ret.num = (int)x;
    simplify(ret);
    return ret;
}

int fraqcmp(Frac a, Frac b) {
    int g1 = gcd(a.den, b.den), g2 = gcd(a.num, b.num);
    if (!g1 || !g2) {
        return 0;
    }
    return b.den / g1 *(a.num / g2) - a.den / g1 *(b.num / g2);
}

Frac add(Frac a, Frac b) {
    int g1 = gcd(a.den, b.den), g2, t;
    if (!g1) {
        return f(1, 0, 0);
    }
    t = b.den / g1 * a.num + a.den / g1 * b.num;
    g2 = gcd(g1, t);
    return f(t / g2, a.den / g1 *(b.den / g2), 0);
}

Frac sub(Frac a, Frac b) {
    return add(a, f(-b.num, b.den, 0));
}

Frac mul(Frac a, Frac b) {
    int t1 = gcd(a.den, b.num), t2 = gcd(a.num, b.den);
    if (!t1 || !t2) {
        return f(1, 1, 0);
    }
    return f(a.num / t2 *(b.num / t1), a.den / t1 *(b.den / t2), 0);
}

Frac div(Frac a, Frac b) {
    return mul(a, f(b.den, b.num, 0));
}
```

## 9.2 日期

```
int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

```

```cpp
class Date {
public:
    //判闰年
    inline static bool isLeap(int year) {
        return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
    }

    int year, month, day;

    //判合法性
    inline bool isLegal() const {
        if (month <= 0 || month > 12) {
            return false;
        }
        if (month == 2) {
            return day > 0 && day <= 28 + isLeap(year);
        }
        return day > 0 && day <= days[month - 1];
    }

    //比较日期大小
    inline int compareTo(const Date &other) const {
        if (year != other.year) {
            return year - other.year;
        }
        if (month != other.month) {
            return month - other.month;
        }
        return day - other.day;
    }

    //返回指定日期是星期几  0 (Sunday) ... 6 (Saturday)
    inline int toWeekday() const {
        int tm = month >= 3 ? (month - 2) : (month + 10);
        int ty = month >= 3 ? year : (year - 1);
        return (ty + ty / 4 - ty / 100 + ty / 400 + (int)(2.6 * tm - 0.2) + day) % 7;
    }

    //日期转天数偏移
    inline int toInt() const {
        int ret = year * 365 + (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400;
        days[1] += isLeap(year);
        for (int i = 0; i < month - 1; ret += days[i++]);
        days[1] = 28;
        return ret + day;
    }

    //天数偏移转日期
    inline void fromInt(int a) {
        year = a / 146097 * 400;
        for (a %= 146097; a >= 365 + isLeap(year); a -= 365 + isLeap(year), year++);
        days[1] += isLeap(year);
        for (month = 1; a >= days[month - 1]; a -= days[month - 1], month++);
        days[1] = 28;
```

```
        day = a + 1;
    }
};
```

## 9.3 矩阵

```
#include <cmath>
const int MAXN = 100;

#define zero(x) (fabs(x) < 1e-10)

struct mat {
    int n, m;
    double data[MAXN][MAXN];
};

bool mul(mat &c, const mat &a, const mat &b) {
    int i, j, k;
    if (a.m != b.n) {
        return false;
    }
    c.n = a.n;
    c.m = b.m;
    for (i = 0; i < c.n; i++) {
        for (j = 0; j < c.m; j++) {
            for (c.data[i][j] = k = 0; k < a.m; k++) {
                c.data[i][j] += a.data[i][k] * b.data[k][j];
            }
        }
    }
    return true;
}

bool inv(mat &a) {
    int i, j, k, is[MAXN], js[MAXN];
    double t;
    if (a.n != a.m) {
        return false;
    }
    for (k = 0; k < a.n; k++) {
        for (t = 0, i = k; i < a.n; i++) {
            for (j = k; j < a.n; j++) {
                if (fabs(a.data[i][j]) > t) {
                    t = fabs(a.data[is[k] = i][js[k] = j]);
                }
            }
        }
        if (zero(t)) {
            return false;
        }
        if (is[k] != k) {
            for (j = 0; j < a.n; j++) {
                t = a.data[k][j];
                a.data[k][j] = a.data[is[k]][j];
                a.data[is[k]][j] = t;
            }
        }
```

```
        if (js[k] != k) {
            for (i = 0; i < a.n; i++) {
                t = a.data[i][k];
                a.data[i][k] = a.data[i][js[k]];
                a.data[i][js[k]] = t;
            }
        }
        a.data[k][k] = 1 / a.data[k][k];
        for (j = 0; j < a.n; j++) {
            if (j != k) {
                a.data[k][j] *= a.data[k][k];
            }
        }
        for (i = 0; i < a.n; i++) {
            if (i != k) {
                for (j = 0; j < a.n; j++) {
                    if (j != k) {
                        a.data[i][j] -= a.data[i][k] * a.data[k][j];
                    }
                }
            }
        }
        for (i = 0; i < a.n; i++) {
            if (i != k) {
                a.data[i][k] *= -a.data[k][k];
            }
        }
    }
    for (k = a.n - 1; k >= 0; k--) {
        for (j = 0; j < a.n; j++) {
            if (js[k] != k) {
                t = a.data[k][j];
                a.data[k][j] = a.data[js[k]][j];
                a.data[js[k]][j] = t;
            }
        }
        for (i = 0; i < a.n; i++) {
            if (is[k] != k) {
                t = a.data[i][k];
                a.data[i][k] = a.data[i][is[k]];
                a.data[i][is[k]] = t;
            }
        }
    }
    return true;
}

double det(const mat &a) {
    int i, j, k, sign = 0;
    double b[MAXN][MAXN], ret = 1, t;
    if (a.n != a.m) {
        return 0;
    }
    for (i = 0; i < a.n; i++) {
        for (j = 0; j < a.m; j++) {
            b[i][j] = a.data[i][j];
        }
    }
```

```
    for (i = 0; i < a.n; i++) {
        if (zero(b[i][i])) {
            for (j = i + 1; j < a.n; j++) {
                if (!zero(b[j][i])) {
                    break;
                }
            }
            if (j == a.n) {
                return 0;
            }
            for (k = i; k < a.n; k++) {
                t = b[i][k], b[i][k] = b[j][k], b[j][k] = t;
            }
            sign++;
        }
        ret *= b[i][i];
        for (k = i + 1; k < a.n; k++) {
            b[i][k] /= b[i][i];
        }
        for (j = i + 1; j < a.n; j++) {
            for (k = i + 1; k < a.n; k++) {
                b[j][k] -= b[j][i] * b[i][k];
            }
        }
    }
    if (sign & 1) {
        ret = -ret;
    }
    return ret;
}
```

## 9.4 awt 基本应用

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.awt.geom.*;

// 计算多个矩形的并，并输出各个矩形面积之和除以矩形并的面积
// 这些矩形的边可以不平行于坐标轴

public class Main {
    private static double calcArea(Area area) {
        double ret = 0;
        PathIterator it = area.getPathIterator(null);
        double[] old = new double[6], now = new double[2], head = null;
        while (!it.isDone()) {
            switch (it.currentSegment(old)) {
            case PathIterator.SEG_MOVETO:
            case PathIterator.SEG_LINETO:
                if (head == null) {
                    head = new double[2];
                    head[0] = old[0];
                    head[1] = old[1];
                } else {
```

```
                    ret += now[0] * old[1] - now[1] * old[0];
                }
                now[0] = old[0];
                now[1] = old[1];
                break;
            case PathIterator.SEG_CLOSE:
                ret += now[0] * head[1] - now[1] * head[0];
                head = null;
                break;
            }
            it.next();
        }
        return ret;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Area tmpArea, combined = new Area();
        int n = sc.nextInt();
        double total = 0, a, b;
        for (int i = 0; i < n; i++) {
            Path2D.Double p = new Path2D.Double();
            a = sc.nextDouble();
            b = sc.nextDouble();
            p.moveTo(a, b);
            for (int j = 0; j < 3; j++) {
                a = sc.nextDouble();
                b = sc.nextDouble();
                p.lineTo(a, b);
            }
            p.closePath();
            tmpArea = new Area(p);
            combined.add(tmpArea);
            total += calcArea(tmpArea);
        }
        System.out.println(total / calcArea(combined));
    }
}
```

# Chapter 10

# 附录

## 10.1 应用

### 10.1.1 N 皇后构造解

```
//N 皇后构造解,n>=4

void even1(int n, int *p) {
    int i;
    for (i = 1; i <= n / 2; i++) {
        p[i - 1] = 2 * i;
    }
    for (i = n / 2 + 1; i <= n; i++) {
        p[i - 1] = 2 * i - n - 1;
    }
}

void even2(int n, int *p) {
    int i;
    for (i = 1; i <= n / 2; i++) {
        p[i - 1] = (2 * i + n / 2 - 3) % n + 1;
    }
    for (i = n / 2 + 1; i <= n; i++) {
        p[i - 1] = n - (2 *(n - i + 1) + n / 2 - 3) % n;
    }
}

void generate(int, int *);

void odd(int n, int *p) {
    generate(n - 1, p);
    p[n - 1] = n;
}

void generate(int n, int *p) {
    if (n & 1) {
        odd(n, p);
    } else if (n % 6 != 2) {
        even1(n, p);
    } else {
        even2(n, p);
    }
```

```
}
```

### 10.1.2 大数 (整数类封装)

```cpp
// 不推荐使用，最好用 Java

#include <iostream>
#include <cstring>
using namespace std;

#define DIGIT   4
#define DEPTH   10000
#define MAX     100
typedef int bignum_t[MAX+1];

int read(bignum_t a,istream &is=cin) {
    char buf[MAX*DIGIT+1],ch;
    int i,j;
    memset((void *)a,0,sizeof(bignum_t));
    if (!(is>>buf)) return 0;
    for (a[0]=strlen(buf),i=a[0]/2-1; i>=0; i--)
        ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
    for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf); j<a[0]*DIGIT; buf[j++]='0');
    for (i=1; i<=a[0]; i++)
        for (a[i]=0,j=0; j<DIGIT; j++)
            a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
    for (; !a[a[0]]&&a[0]>1; a[0]--);
    return 1;
}

void write(const bignum_t a,ostream &os=cout) {
    int i,j;
    for (os<<a[i=a[0]],i--; i; i--)
        for (j=DEPTH/10; j; j/=10)
            os<<a[i]/j%10;
}

int comp(const bignum_t a,const bignum_t b) {
    int i;
    if (a[0]!=b[0])
        return a[0]-b[0];
    for (i=a[0]; i; i--)
        if (a[i]!=b[i])
            return a[i]-b[i];
    return 0;
}

int comp(const bignum_t a,const int b) {
    int c[12]= {1};
    for (c[1]=b; c[c[0]]>=DEPTH; c[c[0]+1]=c[c[0]]/DEPTH,c[c[0]]%=DEPTH,c[0]++);
    return comp(a,c);
}

int comp(const bignum_t a,const int c,const int d,const bignum_t b) {
    int i,t=0,O=-DEPTH*2;
    if (b[0]-a[0]<d&&c)
        return 1;
```

```
    for (i=b[0]; i>d; i--) {
        t=t*DEPTH+a[i-d]*c-b[i];
        if (t>0) return 1;
        if (t<O) return 0;
    }
    for (i=d; i; i--) {
        t=t*DEPTH-b[i];
        if (t>0) return 1;
        if (t<O) return 0;
    }
    return t>0;
}

void add(bignum_t a,const bignum_t b) {
    int i;
    for (i=1; i<=b[0]; i++)
        if ((a[i]+=b[i])>=DEPTH)
            a[i]-=DEPTH,a[i+1]++;
    if (b[0]>=a[0])
        a[0]=b[0];
    else
        for (; a[i]>=DEPTH&&i<a[0]; a[i]-=DEPTH,i++,a[i]++);
    a[0]+=(a[a[0]+1]>0);
}

void add(bignum_t a,const int b) {
    int i=1;
    for (a[1]+=b; a[i]>=DEPTH&&i<a[0]; a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);
    for (; a[a[0]]>=DEPTH; a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
}

void sub(bignum_t a,const bignum_t b) {
    int i;
    for (i=1; i<=b[0]; i++)
        if ((a[i]-=b[i])<0)
            a[i+1]--,a[i]+=DEPTH;
    for (; a[i]<0; a[i]+=DEPTH,i++,a[i]--);
    for (; !a[a[0]]&&a[0]>1; a[0]--);
}

void sub(bignum_t a,const int b) {
    int i=1;
    for (a[1]-=b; a[i]<0; a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,\
i++);
    for (; !a[a[0]]&&a[0]>1; a[0]--);
}

void sub(bignum_t a,const bignum_t b,const int c,const int d) {
    int i,O=b[0]+d;
    for (i=1+d; i<=O; i++)
        if ((a[i]-=b[i-d]*c)<0)
            a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH;
    for (; a[i]<0; a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
    for (; !a[a[0]]&&a[0]>1; a[0]--);
}

void mul(bignum_t c,const bignum_t a,const bignum_t b) {
    int i,j;
```

```
    memset((void *)c,0,sizeof(bignum_t));
    for (c[0]=a[0]+b[0]-1,i=1; i<=a[0]; i++)
        for (j=1; j<=b[0]; j++)
            if ((c[i+j-1]+=a[i]*b[j])>=DEPTH)
                c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;
    for (c[0]+=(c[c[0]+1]>0); !c[c[0]]&&c[0]>1; c[0]--);
}

void mul(bignum_t a,const int b) {
    int i;
    for (a[1]*=b,i=2; i<=a[0]; i++) {
        a[i]*=b;
        if (a[i-1]>=DEPTH)
            a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;
    }
    for (; a[a[0]]>=DEPTH; a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
    for (; !a[a[0]]&&a[0]>1; a[0]--);
}

void mul(bignum_t b,const bignum_t a,const int c,const int d) {
    int i;
    memset((void *)b,0,sizeof(bignum_t));
    for (b[0]=a[0]+d,i=d+1; i<=b[0]; i++)
        if ((b[i]+=a[i-d]*c)>=DEPTH)
            b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;
    for (; b[b[0]+1]; b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);
    for (; !b[b[0]]&&b[0]>1; b[0]--);
}

void div(bignum_t c,bignum_t a,const bignum_t b) {
    int h,l,m,i;
    memset((void *)c,0,sizeof(bignum_t));
    c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;
    for (i=c[0]; i; sub(a,b,c[i]=m,i-1),i--)
        for (h=DEPTH-1,l=0,m=(h+l+1)>>1; h>l; m=(h+l+1)>>1)
            if (comp(b,m,i-1,a)) h=m-1;
            else l=m;
    for (; !c[c[0]]&&c[0]>1; c[0]--);
    c[0]=c[0]>1?c[0]:1;
}

void div(bignum_t a,const int b,int &c) {
    int i;
    for (c=0,i=a[0]; i; c=c*DEPTH+a[i],a[i]=c/b,c%=b,i--);
    for (; !a[a[0]]&&a[0]>1; a[0]--);
}

void sqrt(bignum_t b,bignum_t a) {
    int h,l,m,i;
    memset((void *)b,0,sizeof(bignum_t));
    for (i=b[0]=(a[0]+1)>>1; i; sub(a,b,m,i-1),b[i]+=m,i--)
        for (h=DEPTH-1,l=0,b[i]=m=(h+l+1)>>1; h>l; b[i]=m=(h+l+1)>>1)
            if (comp(b,m,i-1,a)) h=m-1;
            else l=m;
    for (; !b[b[0]]&&b[0]>1; b[0]--);
    for (i=1; i<=b[0]; b[i++]>>=1);
}
```

```
int length(const bignum_t a) {
    int t,ret;
    for (ret=(a[0]-1)*DIGIT,t=a[a[0]]; t; t/=10,ret++);
    return ret>0?ret:1;
}

int digit(const bignum_t a,const int b) {
    int i,ret;
    for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT; i; ret/=10,i--);
    return ret%10;
}

int zeronum(const bignum_t a) {
    int ret,t;
    for (ret=0; !a[ret+1]; ret++);
    for (t=a[ret+1],ret*=DIGIT; !(t%10); t/=10,ret++);
    return ret;
}

void comp(int *a,const int l,const int h,const int d) {
    int i,j,t;
    for (i=l; i<=h; i++)
        for (t=i,j=2; t>1; j++)
            while (!(t%j))
                a[j]+=d,t/=j;
}

void convert(int *a,const int h,bignum_t b) {
    int i,j,t=1;
    memset(b,0,sizeof(bignum_t));
    for (b[0]=b[1]=1,i=2; i<=h; i++)
        if (a[i])
            for (j=a[i]; j; t*=i,j--)
                if (t*i>DEPTH)
                    mul(b,t),t=1;
    mul(b,t);
}

void combination(bignum_t a,int m,int n) {
    int *t=new int[m+1];
    memset((void *)t,0,sizeof(int)*(m+1));
    comp(t,n+1,m,1);
    comp(t,2,m-n,-1);
    convert(t,m,a);
    delete []t;
}

void permutation(bignum_t a,int m,int n) {
    int i,t=1;
    memset(a,0,sizeof(bignum_t));
    a[0]=a[1]=1;
    for (i=m-n+1; i<=m; t*=i++)
        if (t*i>DEPTH)
            mul(a,t),t=1;
    mul(a,t);
}

#define SGN(x) ((x)>0?1:((x)<0?-1:0))
```

```cpp
#define ABS(x) ((x)>0?(x):-(x))

int read(bignum_t a,int &sgn,istream &is=cin) {
    char str[MAX*DIGIT+2],ch,*buf;
    int i,j;
    memset((void *)a,0,sizeof(bignum_t));
    if (!(is>>str)) return 0;
    buf=str,sgn=1;
    if (*buf=='-') sgn=-1,buf++;
    for (a[0]=strlen(buf),i=a[0]/2-1; i>=0; i--)
        ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
    for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf); j<a[0]*DIGIT; buf[j++]='0');
    for (i=1; i<=a[0]; i++)
        for (a[i]=0,j=0; j<DIGIT; j++)
            a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
    for (; !a[a[0]]&&a[0]>1; a[0]--);
    if (a[0]==1&&!a[1]) sgn=0;
    return 1;
}

struct bignum {
    bignum_t num;
    int sgn;
public:
    inline bignum() {
        memset(num,0,sizeof(bignum_t));
        num[0]=1;
        sgn=0;
    }
    inline int operator!() {
        return num[0]==1&&!num[1];
    }
    inline bignum &operator=(const bignum &a) {
        memcpy(num,a.num,sizeof(bignum_t));
        sgn=a.sgn;
        return *this;
    }
    inline bignum &operator=(const int a) {
        memset(num,0,sizeof(bignum_t));
        num[0]=1;
        sgn=SGN(a);
        add(num,sgn*a);
        return *this;
    };
    inline bignum &operator+=(const bignum &a) {
        if(sgn==a.sgn)add(num,a.num);
        else if(sgn&&a.sgn) {
            int ret=comp(num,a.num);
            if(ret>0)sub(num,a.num);
            else if(ret<0) {
                bignum_t t;
                memcpy(t,num,sizeof(bignum_t));
                memcpy(num,a.num,sizeof(bignum_t));
                sub(num,t);
                sgn=a.sgn;
            } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
        } else if(!sgn)memcpy(num,a.num,sizeof(bignum_t)),sgn=a.sgn;
        return *this;
```

```
    }
    inline bignum &operator+=(const int a) {
        if(sgn*a>0)add(num,ABS(a));
        else if(sgn&&a) {
            int ret=comp(num,ABS(a));
            if(ret>0)sub(num,ABS(a));
            else if(ret<0) {
                bignum_t t;
                memcpy(t,num,sizeof(bignum_t));
                memset(num,0,sizeof(bignum_t));
                num[0]=1;
                add(num,ABS(a));
                sgn=-sgn;
                sub(num,t);
            } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
        } else if(!sgn)sgn=SGN(a),add(num,ABS(a));
        return *this;
    }
    inline bignum operator+(const bignum &a) {
        bignum ret;
        memcpy(ret.num,num,sizeof(bignum_t));
        ret.sgn=sgn;
        ret+=a;
        return ret;
    }
    inline bignum operator+(const int a) {
        bignum ret;
        memcpy(ret.num,num,sizeof(bignum_t));
        ret.sgn=sgn;
        ret+=a;
        return ret;
    }
    inline bignum &operator-=(const bignum &a) {
        if(sgn*a.sgn<0)add(num,a.num);
        else if(sgn&&a.sgn) {
            int ret=comp(num,a.num);
            if(ret>0)sub(num,a.num);
            else if(ret<0) {
                bignum_t t;
                memcpy(t,num,sizeof(bignum_t));
                memcpy(num,a.num,sizeof(bignum_t));
                sub(num,t);
                sgn=-sgn;
            } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
        } else if(!sgn)add(num,a.num),sgn=-a.sgn;
        return *this;
    }
    inline bignum &operator-=(const int a) {
        if(sgn*a<0)add(num,ABS(a));
        else if(sgn&&a) {
            int ret=comp(num,ABS(a));
            if(ret>0)sub(num,ABS(a));
            else if(ret<0) {
                bignum_t t;
                memcpy(t,num,sizeof(bignum_t));
                memset(num,0,sizeof(bignum_t));
                num[0]=1;
                add(num,ABS(a));
```

```
                sub(num,t);
                sgn=-sgn;
            } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
        } else if(!sgn)sgn=-SGN(a),add(num,ABS(a));
        return *this;
    }
    inline bignum operator-(const bignum &a) {
        bignum ret;
        memcpy(ret.num,num,sizeof(bignum_t));
        ret.sgn=sgn;
        ret-=a;
        return ret;
    }
    inline bignum operator-(const int a) {
        bignum ret;
        memcpy(ret.num,num,sizeof(bignum_t));
        ret.sgn=sgn;
        ret-=a;
        return ret;
    }
    inline bignum &operator*=(const bignum &a) {
        bignum_t t;
        mul(t,num,a.num);
        memcpy(num,t,sizeof(bignum_t));
        sgn*=a.sgn;
        return *this;
    }
    inline bignum &operator*=(const int a) {
        mul(num,ABS(a));
        sgn*=SGN(a);
        return *this;
    }
    inline bignum operator*(const bignum &a) {
        bignum ret;
        mul(ret.num,num,a.num);
        ret.sgn=sgn*a.sgn;
        return ret;
    }
    inline bignum operator*(const int a) {
        bignum ret;
        memcpy(ret.num,num,sizeof(bignum_t));
        mul(ret.num,ABS(a));
        ret.sgn=sgn*SGN(a);
        return ret;
    }
    inline bignum &operator/=(const bignum &a) {
        bignum_t t;
        div(t,num,a.num);
        memcpy(num,t,sizeof(bignum_t));
        sgn=(num[0]==1&&!num[1])?0:sgn*a.sgn;
        return *this;
    }
    inline bignum &operator/=(const int a) {
        int t;
        div(num,ABS(a),t);
        sgn=(num[0]==1&&!num[1])?0:sgn*SGN(a);
        return *this;
    }
```

```
    inline bignum operator/(const bignum &a) {
        bignum ret;
        bignum_t t;
        memcpy(t,num,sizeof(bignum_t));
        div(ret.num,t,a.num);
        ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*a.sgn;
        return ret;
    }
    inline bignum operator/(const int a) {
        bignum ret;
        int t;
        memcpy(ret.num,num,sizeof(bignum_t));
        div(ret.num,ABS(a),t);
        ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*SGN(a);
        return ret;
    }
    inline bignum &operator%=(const bignum &a) {
        bignum_t t;
        div(t,num,a.num);
        if (num[0]==1&&!num[1])sgn=0;
        return *this;
    }
    inline int operator%=(const int a) {
        int t;
        div(num,ABS(a),t);
        memset(num,0,sizeof(bignum_t));
        num[0]=1;
        add(num,t);
        return t;
    }
    inline bignum operator%(const bignum &a) {
        bignum ret;
        bignum_t t;
        memcpy(ret.num,num,sizeof(bignum_t));
        div(t,ret.num,a.num);
        ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn;
        return ret;
    }
    inline int operator%(const int a) {
        bignum ret;
        int t;
        memcpy(ret.num,num,sizeof(bignum_t));
        div(ret.num,ABS(a),t);
        memset(ret.num,0,sizeof(bignum_t));
        ret.num[0]=1;
        add(ret.num,t);
        return t;
    }
    inline bignum &operator++() {
        *this+=1;
        return *this;
    }
    inline bignum &operator--() {
        *this-=1;
        return *this;
    };
    inline int operator>(const bignum &a) {
        return sgn>0?(a.sgn>0?comp(num,a.num)>0:1):(sgn<0?(a.sgn<0?comp(num,a.num)<0:0):\
```

```
a.sgn<0);
    }
    inline int operator>(const int a) {
        return sgn>0?(a>0?comp(num,a)>0:1):(sgn<0?(a<0?comp(num,-a)<0:0):a<0);
    }
    inline int operator>=(const bignum &a) {
        return sgn>0?(a.sgn>0?comp(num,a.num)>=0:1):(sgn<0?(a.sgn<0?comp(num,a.num)<=0:0\
):a.sgn<=0);
    }
    inline int operator>=(const int a) {
        return sgn>0?(a>0?comp(num,a)>=0:1):(sgn<0?(a<0?comp(num,-a)<=0:0):a<=0);
    }
    inline int operator<(const bignum &a) {
        return sgn<0?(a.sgn<0?comp(num,a.num)>0:1):(sgn>0?(a.sgn>0?comp(num,a.num)<0:0):\
a.sgn>0);
    }
    inline int operator<(const int a) {
        return sgn<0?(a<0?comp(num,-a)>0:1):(sgn>0?(a>0?comp(num,a)<0:0):a>0);
    }
    inline int operator<=(const bignum &a) {
        return sgn<0?(a.sgn<0?comp(num,a.num)>=0:1):(sgn>0?(a.sgn>0?comp(num,a.num)<=0:0\
):a.sgn>=0);
    }
    inline int operator<=(const int a) {
        return sgn<0?(a<0?comp(num,-a)>=0:1):(sgn>0?(a>0?comp(num,a)<=0:0):a>=0);
    }
    inline int operator==(const bignum &a) {
        return (sgn==a.sgn)?!comp(num,a.num):0;
    }
    inline int operator==(const int a) {
        return (sgn*a>=0)?!comp(num,ABS(a)):0;
    }
    inline int operator!=(const bignum &a) {
        return (sgn==a.sgn)?comp(num,a.num):1;
    }
    inline int operator!=(const int a) {
        return (sgn*a>=0)?comp(num,ABS(a)):1;
    }
    inline int operator[](const int a) {
        return digit(num,a);
    }
    friend inline istream &operator>>(istream &is,bignum &a) {
        read(a.num,a.sgn,is);
        return is;
    }
    friend inline ostream &operator<<(ostream &os,const bignum &a) {
        if(a.sgn<0)os<<'-';
        write(a.num,os);
        return os;
    }
    friend inline bignum sqrt(const bignum &a) {
        bignum ret;
        bignum_t t;
        memcpy(t,a.num,sizeof(bignum_t));
        sqrt(ret.num,t);
        ret.sgn=ret.num[0]!=1||ret.num[1];
        return ret;
    }
```

```
    friend inline bignum sqrt(const bignum &a,bignum &b) {
        bignum ret;
        memcpy(b.num,a.num,sizeof(bignum_t));
        sqrt(ret.num,b.num);
        ret.sgn=ret.num[0]!=1||ret.num[1];
        b.sgn=b.num[0]!=1||ret.num[1];
        return ret;
    }
    inline int length() {
        return ::length(num);
    }
    inline int zeronum() {
        return ::zeronum(num);
    }
    inline bignum C(const int m,const int n) {
        combination(num,m,n);
        sgn=1;
        return *this;
    }
    inline bignum P(const int m,const int n) {
        permutation(num,m,n);
        sgn=1;
        return *this;
    }
};
```

### 10.1.3 幻方构造

```
//幻方构造 (L!=2)
const int MAXN = 100;

void dllb(int L, int si, int sj, int sn, int d[][MAXN]) {
    int n, i = 0, j = L / 2;
    for (n = 1; n <= L * L; n++) {
        d[i + si][j + sj] = n + sn;
        if (n % L) {
            i = (i) ? (i - 1): (L - 1);
            j = (j == L - 1) ? 0 : (j + 1);
        } else {
            i = (i == L - 1) ? 0 : (i + 1);
        }
    }
}

void magicOdd(int L, int d[][MAXN]) {
    dllb(L, 0, 0, 0, d);
}

void magic4k(int L, int d[][MAXN]) {
    int i, j;
    for (i = 0; i < L; i++) {
        for (j = 0; j < L; j++) {
            d[i][j] = ((i % 4 == 0 || i % 4 == 3) && (j % 4 == 0 || j % 4 == 3) || (i % \
4 == 1 || i % 4 == 2) && (j % 4 == 1 || j % 4 == 2)) ? (L * L - (i * L + j)) : (i * L + \
j + 1);
        }
    }
```

```
}

void magicOther(int L, int d[][MAXN]) {
    int i, j, t;
    dllb(L / 2, 0, 0, 0, d);
    dllb(L / 2, L / 2, L / 2, L * L / 4, d);
    dllb(L / 2, 0, L / 2, L * L / 2, d);
    dllb(L / 2, L / 2, 0, L * L / 4 * 3, d);
    for (i = 0; i < L / 2; i++) {
        for (j = 0; j < L / 4; j++) {
            if (i != L / 4 || j) {
                t = d[i][j];
                d[i][j] = d[i + L / 2][j];
                d[i + L / 2][j] = t;
            }
        }
    }
    t = d[L / 4][L / 4];
    d[L / 4][L / 4] = d[L / 4+L / 2][L / 4];
    d[L / 4+L / 2][L / 4] = t;
    for (i = 0; i < L / 2; i++) {
        for (j = L - L / 4+1; j < L; j++) {
            t = d[i][j];
            d[i][j] = d[i + L / 2][j];
            d[i + L / 2][j] = t;
        }
    }
}

void generate(int L, int d[][MAXN]) {
    if (L % 2) {
        magicOdd(L, d);
    } else if (L % 4 == 0) {
        magic4k(L, d);
    } else {
        magicOther(L, d);
    }
}
```

### 10.1.4 最大子串匹配

```
//最大子串匹配, 复杂度 O(mn)
//返回最大匹配值, 传入两个串和串的长度, 重载返回一个最大匹配
//注意做字符串匹配是串末的'\0' 没有置!
//可更改元素类型, 更换匹配函数和匹配价值函数
#include <cstring>
#include <algorithm>
using namespace std;

const int MAXN = 100;

#define _match(a, b) ((a) == (b))
#define _value(a, b) 1

template <class elemType>
```

```
int strMatch(int m, const elemType *a, int n, const elemType *b) {
    int match[MAXN + 1][MAXN + 1], i, j;
    memset(match, 0, sizeof(match));
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            match[i + 1][j + 1] = max(max(match[i][j + 1], match[i + 1][j]), (match[i][j\
] + _value(a[i], b[i])) * _match(a[i], b[j]));
        }
    }
    return match[m][n];
}

template <class elemType>
int strMatch(int m, const elemType *a, int n, const elemType *b, elemType *ret) {
    int match[MAXN + 1][MAXN + 1], last[MAXN + 1][MAXN + 1], i, j, t;
    memset(match, 0, sizeof(match));
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            match[i + 1][j + 1] = (match[i][j + 1] > match[i + 1][j] ? match[i][j + 1] :\
 match[i + 1][j]);
            last[i + 1][j + 1] = (match[i][j + 1] > match[i + 1][j] ? 3 : 1);
            if ((t = (match[i][j] + _value(a[i], b[i])) * _match(a[i], b[j]))> match[i +\
 1][j + 1]) {
                match[i + 1][j + 1] = t;
                last[i + 1][j + 1] = 2;
            }
        }
    }
    for (; match[i][j]; i -= (last[t = i][j] > 1), j -= (last[t][j] < 3)) {
        ret[match[i][j] - 1] = (last[i][j] < 3 ? a[i - 1]: b[j - 1]);
    }
    return match[m][n];
}
```

### 10.1.5 最大子段和

```
//求最大子段和，复杂度 O(n)
//传入串长 n 和内容 list[]
//返回最大子段和，重载返回子段位置 (maxsum=list[start]+...+list[end])
//可更改元素类型
template <class elemType>
elemType maxsum(int n, const elemType *list) {
    elemType ret, sum = 0;
    int i;
    for (ret = list[i = 0]; i < n; i++) {
        sum = (sum > 0 ? sum : 0) + list[i];
        ret = (sum > ret ? sum : ret);
    }
    return ret;
}

template <class elemType>
elemType maxsum(int n, const elemType *list, int &start, int &end) {
    elemType ret, sum = 0;
    int s, i;
```

```
    for (ret = list[start = end = s = i = 0]; i < n; i++, s = (sum > 0 ? s : i)) {
        if ((sum = (sum > 0 ? sum : 0) + list[i])> ret) {
            ret = sum;
            start = s;
            end = i;
        }
    }
    return ret;
}
```

### 10.1.6 第 k 元素

```
//一般可用 STL 的 kth_element()

//取第 k 个元素,k=0..n-1
//平均复杂度 O(n)
//注意 a[] 中的顺序被改变
#define _cp(a, b) ((a) < (b))

template <class elemType>
elemType kthElement(int n, const elemType *a, int k) {
    elemType t, key;
    int left = 0, r = n - 1, i, j;
    while (left < r) {
        for (key = a[((i = left - 1) + (j = r + 1)) >> 1]; i < j;) {
            for (j--; _cp(key, a[j]); j--);
            for (i++; _cp(a[i], key); i++);
            if (i<j) {
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
        if (k > j) {
            left = j + 1;
        } else {
            r = j;
        }
    }
    return a[k];
}
```

### 10.1.7 骰子

```
//Author: t__nt
//骰子的基本操作
//展开后对应的标号
//   3
//1 2 6 5
//   4
#include<cstdio>
```

```
//通过上表面和前面，得出右侧面
const int getFace[7][7]=//[upward][forward]
    //0 1 2 3 4 5 6
{   {0,0,0,0,0,0,0}    //0
    ,{0,0,3,5,2,4,0}    //1
    ,{0,4,0,1,6,0,3}    //2
    ,{0,2,6,0,0,1,5}    //3
    ,{0,5,1,0,0,6,2}    //4
    ,{0,3,0,6,1,0,4}    //5
    ,{0,0,4,2,5,3,0}
};  //6
//对面的对应标号
const int oppo[7]= {0,6,5,4,3,2,1};

class Dice {
public:
    int up,forward,right;

    Dice() {}

    Dice(int a,int b):up(a),forward(b) {
        right=getFace[a][b];
    }

    Dice(int a,int b,int c):up(a),forward(b),right(c) {}

    void goLeft() {
        int a=up,b=forward,c=right;
        up=c;
        forward=b;
        rigth=oppo[a];
    }

    void goRight() {
        int a=up,b=forward,c=right;
        up=oppo[c];
        forward=b;
        rigth=a;
    }

    void goUp() {
        int a=up,b=forward,c=right;
        up=b;
        forward=oppo[a];
        rigth=c;
    }

    void goDown(Pos &tmppos,Pos &pos) {
        int a=up,b=forward,c=right;
        up=oppo[b];
        forward=a;
        rigth=c;
```

```
    }

};
```

# 10.2 算法描述

## 10.2.1 弦图与区间图

```
// 弦图与区间图 By 猛犸也钻地 @ 2012.09.13

/* 相关定义 //
1. 子图：原图点集的子集 + 原图边集的子集
2. 诱导子图：原图点集的子集 + 这些点在原图中所连出的边集
3. 团：原图的一个子图，且是完全图
4. 极大团：此团不是其他团的子集
5. 最大团：点数最多的团 -> 团数
6. 最小染色：用最少的颜色给点染色使相邻点颜色不同 -> 色数
7. 最大独立集：原图点集的子集，任意两点在原图中没有边相连
8. 最小团覆盖：用最少个数的团覆盖所有的点
   推论 -> 团数<= 色数，最大独立集数<= 最小团覆盖数
9. 弦：连接环中不相邻的两个点的边
10. 弦图：图中任意长度大于 3 的环都至少有 1 个弦
   推论 -> 弦图的每一个诱导子图一定是弦图
           弦图的任一个诱导子图不同构于 Cn(n>3)
11. 单纯点：记 N(v) 为点 v 相邻点的集合，若 N(v)+{v} 是一个团，则 v 为单纯点
   引理 -> 任何一个弦图都至少有一个单纯点
           不是完全图的弦图至少有两个不相邻的单纯
// 重点内容 //
12. 完美消除序列：点的序列 v1,v2,..,vn，满足 vi 在 {vi,vi+1,..,vn} 中是单纯点
   定理 -> 一个无向图是弦图，当且仅当它有一个完美消除序列
   构造算法 -> 令 cnt[i] 为第 i 个点与多少个已标记的点相邻，初值全为零
               每次选择一个 cnt[i] 最大的结点并打上标记
               标记顺序的逆序则为完美消除序列
   判定算法 -> 对于每个 vi，其出边为 vi1,vi2,..,vik
               然后判断 vi1 与 vi2,vi3,..,vik 是否都相邻
               若存在不相邻的情况，则说明不是完美消除序列
13. 弦图各类算法：
   最小染色算法 -> 按照完美消除序列，从后向前，依次染上可以染的最小颜色
   最大独立集算法 -> 按照完美消除序列，从前向后，能选则选
   最小团覆盖算法 -> 求出最大独立集，记为 {p1,p2,..,pk}
                     则 {N(p1)+{p1},N(p2)+{p2},..,N(pk)+{pk}} 即为解
16. 区间图：坐标轴上的一些区间看作点，任意两个交集非空的区间之间有边
   定理 -> 区间图一定是弦图 */
```

## 10.2.2 生成树

```
// 生成树相关的一些问题 By 猛犸也钻地 @ 2012.02.24

/* 度限制生成树 //
Q: 求一个最小生成树，其中 V0 连接的边不能超过 K 个或只能刚好 K 个
1. 去掉所有和 V0 连接的边，对每个连通分量求最小生成树
2. 如果除去点 V0 外共有 T 个连通分量，且 T>K，无解
3. 于是现在有一个最小 T 度生成树，然后用 dp[V] 计算出该点到 V0 的路径上
   权值最大的边是多少，再枚举和 V0 连接的没有使用过的边，找出一条边
   使得用那条边替换已有的边，增加的权值最小，不停替换直到 V0 出度为 K */
```

```
/* 次小生成树 //
Q: 求一个次小生成树，要求权值之和必须大于等于或严格大于其最小生成树
1. 求最小生成树
2. 找一个根然后 dp，求出每个点往上走 2^L 能到达的祖先是谁，以及
   这段路径上的最大边和次大边 (如果仅要求大于等于的话就不需要次大边)
3. 枚举没有使用过的边，利用上面得到的信息，在 O(logN) 时间内对每条边
   计算出其能够替换的已有的最大和次大边，然后找出最佳替换方式 */

/* 斯坦纳树 //
Q: 求一个包含指定的 K 个特殊点的最小生成树，其他点不一定在树中
1. 用 dp[mask][x] 记录树根在点 x，mask 所对应的特殊点集在树中的最小权值之和
2. 将 dp[][] 初始化为正无穷，只有 dp[1<<i][Ai] 被初始化为 0，Ai 为第 i 个特殊点
3. 先求出所有点对间最短路，然后枚举 mask，依次做两种转移:
3.1. 枚举 x 和 mask 的子集 sub，合并两棵子树
     dp[mask][x]=min(dp[mask][x],dp[sub][x]+dp[mask^sub][x])
3.2. 枚举 x 和 y，计算结点从 y 移动到 x 的花费
     dp[mask][x]=min(dp[mask][x],dp[mask][y]+minDistance(y,x))
     在上面的转移中，也可以把这些点同时放到队列里，用 spfa 更新最短路 */

/* 生成树计数 //
Q: 给定一个无权的无向图 G，求生成树的个数
1. 令矩阵 D[][] 为度数矩阵，其中 D[i][i] 为结点 i 的度，其他位置的值为 0
2. 令矩阵 A[][] 为邻接矩阵，当结点 i 和 j 之间有 x 条边时，D[i][j]=D[j][i]=x
3. 令矩阵 C=D-A，矩阵 C' 为矩阵 C 抽去第 k 行和第 k 列后的一个 n-1 阶的子矩阵
   其中 k 可以任意设定，构造完 C' 后，生成树的个数即为 C' 行列式的值 */
```

### 10.2.3 有向图最小均值环

求强连通图最小均值环的 Karp 算法 By sfiction @ 2016.12.19

图不连通时对每个强联通分量独立做一遍。任意选取一个起点 $s$，设 $F_k(v)$ 为 $s$ 到 $v$ 的恰好包含 $k$ 条边的最短路径，若不存在则 $F_k(v)=\infty$。

$$ans = \lambda^* = \min_{v\in V} \max_{0\le k\le n-1} \left[\frac{F_n(v)-F_k(v)}{n-k}\right]$$

$F_k(v)$ 可以通过 $O(VE)$ 的递推方法求出。

# Chapter 11

# Cheat Sheet

## 11.1 Theoretical Computer Science

## Theoretical Computer Science Cheat Sheet

### Definitions

| | |
|---|---|
| $f(n) = O(g(n))$ | iff $\exists$ positive $c, n_0$ such that $0 \leq f(n) \leq cg(n)\ \forall n \geq n_0$. |
| $f(n) = \Omega(g(n))$ | iff $\exists$ positive $c, n_0$ such that $f(n) \geq cg(n) \geq 0\ \forall n \geq n_0$. |
| $f(n) = \Theta(g(n))$ | iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. |
| $f(n) = o(g(n))$ | iff $\lim_{n\to\infty} f(n)/g(n) = 0$. |
| $\lim\limits_{n\to\infty} a_n = a$ | iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \geq n_0$. |
| $\sup S$ | least $b \in \mathbb{R}$ such that $b \geq s$, $\forall s \in S$. |
| $\inf S$ | greatest $b \in \mathbb{R}$ such that $b \leq s$, $\forall s \in S$. |
| $\liminf\limits_{n\to\infty} a_n$ | $\lim\limits_{n\to\infty} \inf\{a_i \mid i \geq n, i \in \mathbb{N}\}$. |
| $\limsup\limits_{n\to\infty} a_n$ | $\lim\limits_{n\to\infty} \sup\{a_i \mid i \geq n, i \in \mathbb{N}\}$. |
| $\binom{n}{k}$ | Combinations: Size $k$ subsets of a size $n$ set. |
| $\left[{n \atop k}\right]$ | Stirling numbers (1st kind): Arrangements of an $n$ element set into $k$ cycles. |
| $\left\{{n \atop k}\right\}$ | Stirling numbers (2nd kind): Partitions of an $n$ element set into $k$ non-empty sets. |
| $\left\langle{n \atop k}\right\rangle$ | 1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with $k$ ascents. |
| $\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle$ | 2nd order Eulerian numbers. |
| $C_n$ | Catalan Numbers: Binary trees with $n+1$ vertices. |

### Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \neq 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n}\frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

### Identities

**1.** $\displaystyle\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\displaystyle\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\displaystyle\binom{n}{k} = \binom{n}{n-k}$,

**4.** $\displaystyle\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\displaystyle\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

**6.** $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

**8.** $\displaystyle\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\displaystyle\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

**10.** $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

**12.** $\displaystyle\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\displaystyle\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

**14.** $\displaystyle\left[{n \atop 1}\right] = (n-1)!$, **15.** $\displaystyle\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\displaystyle\left[{n \atop n}\right] = 1$, **17.** $\displaystyle\left[{n \atop k}\right] \geq \left\{{n \atop k}\right\}$,

**18.** $\displaystyle\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\displaystyle\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^{n}\left[{n \atop k}\right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

**22.** $\displaystyle\left\langle{n \atop 0}\right\rangle = \left\langle{n \atop n-1}\right\rangle = 1$, **23.** $\displaystyle\left\langle{n \atop k}\right\rangle = \left\langle{n \atop n-1-k}\right\rangle$, **24.** $\displaystyle\left\langle{n \atop k}\right\rangle = (k+1)\left\langle{n-1 \atop k}\right\rangle + (n-k)\left\langle{n-1 \atop k-1}\right\rangle$,

**25.** $\displaystyle\left\langle{0 \atop k}\right\rangle = \begin{cases}1 & \text{if } k = 0,\\ 0 & \text{otherwise}\end{cases}$ **26.** $\displaystyle\left\langle{n \atop 1}\right\rangle = 2^n - n - 1$, **27.** $\displaystyle\left\langle{n \atop 2}\right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

**28.** $\displaystyle x^n = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{x+k}{n}$, **29.** $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $\displaystyle m!\left\{{n \atop m}\right\} = \sum_{k=0}^{n}\left\langle{n \atop k}\right\rangle\binom{k}{n-m}$,

**31.** $\displaystyle\left\langle{n \atop m}\right\rangle = \sum_{k=0}^{n}\left\{{n \atop k}\right\}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\displaystyle\left\langle\!\!\left\langle{n \atop 0}\right\rangle\!\!\right\rangle = 1$, **33.** $\displaystyle\left\langle\!\!\left\langle{n \atop n}\right\rangle\!\!\right\rangle = 0$ for $n \neq 0$,

**34.** $\displaystyle\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle{n-1 \atop k}\right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle{n-1 \atop k-1}\right\rangle\!\!\right\rangle$, **35.** $\displaystyle\sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

**36.** $\displaystyle\left\{{x \atop x-n}\right\} = \sum_{k=0}^{n}\left\langle\!\!\left\langle{n \atop k}\right\rangle\!\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\displaystyle\left\{{n+1 \atop m+1}\right\} = \sum_{k}\binom{n}{k}\left\{{k \atop m}\right\} = \sum_{k=0}^{n}\left\{{k \atop m}\right\}(m+1)^{n-k}$,

## Theoretical Computer Science Cheat Sheet

### Identities Cont.

**38.** $\left[{n+1 \atop m+1}\right] = \sum_k \left[{n \atop k}\right]\binom{k}{m} = \sum_{k=0}^{n} \left[{k \atop m}\right] n^{\underline{n-k}} = n! \sum_{k=0}^{n} \frac{1}{k!}\left[{k \atop m}\right]$, **39.** $\left[{x \atop x-n}\right] = \sum_{k=0}^{n} \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+k}{2n}$,

**40.** $\left\{{n \atop m}\right\} = \sum_k \binom{n}{k}\left\{{k+1 \atop m+1}\right\}(-1)^{n-k}$, **41.** $\left[{n \atop m}\right] = \sum_k \left[{n+1 \atop k+1}\right]\binom{k}{m}(-1)^{m-k}$,

**42.** $\left\{{m+n+1 \atop m}\right\} = \sum_{k=0}^{m} k\left\{{n+k \atop k}\right\}$, **43.** $\left[{m+n+1 \atop m}\right] = \sum_{k=0}^{m} k(n+k)\left[{n+k \atop k}\right]$,

**44.** $\binom{n}{m} = \sum_k \left\{{n+1 \atop k+1}\right\}\left[{k \atop m}\right](-1)^{m-k}$, **45.** $(n-m)!\binom{n}{m} = \sum_k \left[{n+1 \atop k+1}\right]\left\{{k \atop m}\right\}(-1)^{m-k}$, for $n \ge m$,

**46.** $\left\{{n \atop n-m}\right\} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left[{m+k \atop k}\right]$, **47.** $\left[{n \atop n-m}\right] = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}\left\{{m+k \atop k}\right\}$,

**48.** $\left\{{n \atop \ell+m}\right\}\binom{\ell+m}{\ell} = \sum_k \left\{{k \atop \ell}\right\}\left\{{n-k \atop m}\right\}\binom{n}{k}$, **49.** $\left[{n \atop \ell+m}\right]\binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell}\right]\left[{n-k \atop m}\right]\binom{n}{k}$.

### Trees

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

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

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

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

### Recurrences

Master method:
$T(n) = aT(n/b) + f(n), \quad a \ge 1, b > 1$

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

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

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

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

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

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

Substitution (example): Consider the following recurrence

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

Note that $T_i$ is always a power of two. Let $t_i = \log_2 T_i$. Then we have

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

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

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

Substituting we find

$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that $T_i$ has the closed form $T_i = 2^{i2^{i-1}}$. Summing factors (example): Consider the following recurrence

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

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

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

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

$$1\big(T(n) - 3T(n/2) = n\big)$$
$$3\big(T(n/2) - 3T(n/4) = n/2\big)$$
$$\vdots \quad \vdots \quad \vdots$$
$$3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big)$$

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

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

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

$$\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n\left(\frac{c^m - 1}{c-1}\right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_c n} - 1) \\ &= 2n^k - 2n, \end{aligned}$$

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

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

Note that

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

Subtracting we find

$$\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$$

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

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

Example:

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

Multiply and sum:

$$\sum_{i\ge0} g_{i+1}x^i = \sum_{i\ge0} 2g_i x^i + \sum_{i\ge0} x^i.$$

We choose $G(x) = \sum_{i\ge0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i\ge0} x^i.$$

Simplify:

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

Solve for $G(x)$:

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

Expand this using partial fractions:

$$\begin{aligned} G(x) &= x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right) \\ &= x\left(2\sum_{i\ge0} 2^i x^i - \sum_{i\ge0} x^i\right) \\ &= \sum_{i\ge0}(2^{i+1} - 1)x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

## Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

| $i$ | $2^i$ | $p_i$ |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 4 | 3 |
| 3 | 8 | 5 |
| 4 | 16 | 7 |
| 5 | 32 | 11 |
| 6 | 64 | 13 |
| 7 | 128 | 17 |
| 8 | 256 | 19 |
| 9 | 512 | 23 |
| 10 | 1,024 | 29 |
| 11 | 2,048 | 31 |
| 12 | 4,096 | 37 |
| 13 | 8,192 | 41 |
| 14 | 16,384 | 43 |
| 15 | 32,768 | 47 |
| 16 | 65,536 | 53 |
| 17 | 131,072 | 59 |
| 18 | 262,144 | 61 |
| 19 | 524,288 | 67 |
| 20 | 1,048,576 | 71 |
| 21 | 2,097,152 | 73 |
| 22 | 4,194,304 | 79 |
| 23 | 8,388,608 | 83 |
| 24 | 16,777,216 | 89 |
| 25 | 33,554,432 | 97 |
| 26 | 67,108,864 | 101 |
| 27 | 134,217,728 | 103 |
| 28 | 268,435,456 | 107 |
| 29 | 536,870,912 | 109 |
| 30 | 1,073,741,824 | 113 |
| 31 | 2,147,483,648 | 127 |
| 32 | 4,294,967,296 | 131 |

### Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

### General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$$B_0 = 1,\ B_1 = -\tfrac{1}{2},\ B_2 = \tfrac{1}{6},\ B_4 = -\tfrac{1}{30},\ B_6 = \tfrac{1}{42},\ B_8 = -\tfrac{1}{30},\ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number $e$:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$$1,\ \tfrac{3}{2},\ \tfrac{11}{6},\ \tfrac{25}{12},\ \tfrac{137}{60},\ \tfrac{49}{20},\ \tfrac{363}{140},\ \tfrac{761}{280},\ \tfrac{7129}{2520}, \ldots$$

$$\ln n < H_n < \ln n + 1,$$

$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i,j \geq 2 \end{cases}$$

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are $n$ different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all $n$ types is

$$nH_n.$$

### Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then $p$ is the probability density function of $X$. If

$$\Pr[X < a] = P(a),$$

then $P$ is the distribution function of $X$. If $P$ and $p$ both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If $X$ is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x) \Pr[X = x].$$

If $X$ continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events $A$ and $B$:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff $A$ and $B$ are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables $X$ and $Y$:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if $X$ and $Y$ are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$

$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n} \Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \Pr[X_i] + \sum_{k=2}^{n} (-1)^{k+1} \sum_{i_i < \cdots < i_k} \Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$

$$\Pr\left[\big|X - \mathrm{E}[X]\big| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

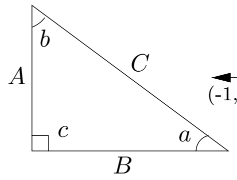
Geometric distribution:

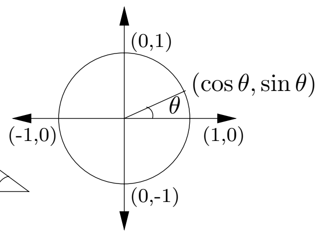
$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

## Theoretical Computer Science Cheat Sheet

### Trigonometry




Pythagorean theorem:
$$C^2 = A^2 + B^2.$$

Definitions:
$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:
$$\tfrac{1}{2}AB, \qquad \frac{AB}{A+B+C}.$$

Identities:
$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2 \sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x+y)\sin(x-y) = \sin^2 x - \sin^2 y,$$
$$\cos(x+y)\cos(x-y) = \cos^2 x - \sin^2 y.$$

Euler's equation:
$$e^{ix} = \cos x + i \sin x, \qquad e^{i\pi} = -1.$$

v2.02 ©1994 by Steve Seiden
sseiden@acm.org
http://www.csc.lsu.edu/~seiden

### Matrices

Multiplication:
$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff $A$ is non-singular.
$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi} \prod_{i=1}^{n} \operatorname{sign}(\pi) a_{i,\pi(i)}.$$

$2 \times 2$ and $3 \times 3$ determinant:
$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ - ceg - fha - ibd. \end{matrix}$$

Permanents:
$$\operatorname{perm} A = \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)}.$$

### Hyperbolic Functions

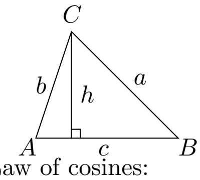
Definitions:
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:
$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \operatorname{sech}^2 x = 1,$$
$$\coth^2 x - \operatorname{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2 \sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2 \tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2 \tfrac{x}{2} = \cosh x + 1.$$

| $\theta$ | $\sin\theta$ | $\cos\theta$ | $\tan\theta$ |
|---|---|---|---|
| $0$ | $0$ | $1$ | $0$ |
| $\frac{\pi}{6}$ | $\frac{1}{2}$ | $\frac{\sqrt{3}}{2}$ | $\frac{\sqrt{3}}{3}$ |
| $\frac{\pi}{4}$ | $\frac{\sqrt{2}}{2}$ | $\frac{\sqrt{2}}{2}$ | $1$ |
| $\frac{\pi}{3}$ | $\frac{\sqrt{3}}{2}$ | $\frac{1}{2}$ | $\sqrt{3}$ |
| $\frac{\pi}{2}$ | $1$ | $0$ | $\infty$ |

. . . in mathematics you don't understand things, you just get used to them.
– J. von Neumann

### More Trig.




Law of cosines:
$$c^2 = a^2 + b^2 - 2ab\cos C.$$

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

Heron's formula:
$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:
$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

**Theoretical Computer Science Cheat Sheet**

## Number Theory

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

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

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

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

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

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

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

Fermat's theorem:

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

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

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

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

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

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

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

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

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

If

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

then

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

Prime numbers:

$$p_n = n \ln n + n \ln\ln n - n + n \frac{\ln\ln n}{\ln n} + O\Big(\frac{n}{\ln n}\Big),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\Big(\frac{n}{(\ln n)^4}\Big).$$

## Graph Theory

Definitions:

| | |
|---|---|
| *Loop* | An edge connecting a vertex to itself. |
| *Directed* | Each edge has a direction. |
| *Simple* | Graph with no loops or multi-edges. |
| *Walk* | A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$. |
| *Trail* | A walk with distinct edges. |
| *Path* | A trail with distinct vertices. |
| *Connected* | A graph where there exists a path between any two vertices. |
| *Component* | A maximal connected subgraph. |
| *Tree* | A connected acyclic graph. |
| *Free tree* | A tree with no root. |
| *DAG* | Directed acyclic graph. |
| *Eulerian* | Graph with a trail visiting each edge exactly once. |
| *Hamiltonian* | Graph with a cycle visiting each vertex exactly once. |
| *Cut* | A set of edges whose removal increases the number of components. |
| *Cut-set* | A minimal cut. |
| *Cut edge* | A size 1 cut. |
| *k-Connected* | A graph connected with the removal of any $k-1$ vertices. |
| *k-Tough* | $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \lvert S \rvert$. |
| *k-Regular* | A graph where all vertices have degree $k$. |
| *k-Factor* | A $k$-regular spanning subgraph. |
| *Matching* | A set of edges, no two of which are adjacent. |
| *Clique* | A set of vertices, all of which are adjacent. |
| *Ind. set* | A set of vertices, none of which are adjacent. |
| *Vertex cover* | A set of vertices which cover all edges. |
| *Planar graph* | A graph which can be embeded in the plane. |
| *Plane graph* | An embedding of a planar graph. |

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

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

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

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

Notation:

| | |
|---|---|
| $E(G)$ | Edge set |
| $V(G)$ | Vertex set |
| $c(G)$ | Number of components |
| $G[S]$ | Induced subgraph |
| $\deg(v)$ | Degree of $v$ |
| $\Delta(G)$ | Maximum degree |
| $\delta(G)$ | Minimum degree |
| $\chi(G)$ | Chromatic number |
| $\chi_E(G)$ | Edge chromatic number |
| $G^c$ | Complement graph |
| $K_n$ | Complete graph |
| $K_{n_1,n_2}$ | Complete bipartite graph |
| $\mathrm{r}(k, \ell)$ | Ramsey number |

## Geometry

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

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

| Cartesian | Projective |
|---|---|
| $(x, y)$ | $(x, y, 1)$ |
| $y = mx + b$ | $(m, -1, b)$ |
| $x = c$ | $(1, 0, -c)$ |

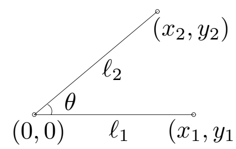
Distance formula, $L_p$ and $L_\infty$ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
$$\big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p},$$
$$\lim_{p \to \infty} \big[|x_1 - x_0|^p + |y_1 - y_0|^p\big]^{1/p}.$$

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

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

Angle formed by three points:




$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

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

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

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

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

## Theoretical Computer Science Cheat Sheet

### $\pi$

Wallis' identity:

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

Brouncker's continued fraction expansion:

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

Gregrory's series:

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

Newton's series:

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

Sharp's series:

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

Euler's series:

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

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

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

### Partial Fractions

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

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

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

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

where

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

For a repeated factor:

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

where

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

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

### Calculus

Derivatives:

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

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

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

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

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

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

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

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

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

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

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

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

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

**29.** $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **30.** $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx}$,

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

Integrals:

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

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

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

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

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

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

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

**Theoretical Computer Science Cheat Sheet**

Calculus Cont.

$$\textbf{15.}\ \int \arccos \tfrac{x}{a} dx = \arccos \tfrac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$$

$$\textbf{16.}\ \int \arctan \tfrac{x}{a} dx = x \arctan \tfrac{x}{a} - \tfrac{a}{2} \ln(a^2 + x^2), \quad a > 0,$$

$$\textbf{17.}\ \int \sin^2(ax)dx = \tfrac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$$

$$\textbf{18.}\ \int \cos^2(ax)dx = \tfrac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$$

$$\textbf{19.}\ \int \sec^2 x\,dx = \tan x,$$

$$\textbf{20.}\ \int \csc^2 x\,dx = -\cot x,$$

$$\textbf{21.}\ \int \sin^n x\,dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x\,dx,$$

$$\textbf{22.}\ \int \cos^n x\,dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x\,dx,$$

$$\textbf{23.}\ \int \tan^n x\,dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x\,dx, \quad n \neq 1,$$

$$\textbf{24.}\ \int \cot^n x\,dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x\,dx, \quad n \neq 1,$$

$$\textbf{25.}\ \int \sec^n x\,dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x\,dx, \quad n \neq 1,$$

$$\textbf{26.}\ \int \csc^n x\,dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x\,dx, \quad n \neq 1,$$

$$\textbf{27.}\ \int \sinh x\,dx = \cosh x, \qquad \textbf{28.}\ \int \cosh x\,dx = \sinh x,$$

$$\textbf{29.}\ \int \tanh x\,dx = \ln|\cosh x|, \quad \textbf{30.}\ \int \coth x\,dx = \ln|\sinh x|, \quad \textbf{31.}\ \int \operatorname{sech} x\,dx = \arctan \sinh x, \quad \textbf{32.}\ \int \operatorname{csch} x\,dx = \ln\left|\tanh \tfrac{x}{2}\right|,$$

$$\textbf{33.}\ \int \sinh^2 x\,dx = \tfrac{1}{4}\sinh(2x) - \tfrac{1}{2}x, \quad \textbf{34.}\ \int \cosh^2 x\,dx = \tfrac{1}{4}\sinh(2x) + \tfrac{1}{2}x, \quad \textbf{35.}\ \int \operatorname{sech}^2 x\,dx = \tanh x,$$

$$\textbf{36.}\ \int \operatorname{arcsinh} \tfrac{x}{a} dx = x \operatorname{arcsinh} \tfrac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$$

$$\textbf{37.}\ \int \operatorname{arctanh} \tfrac{x}{a} dx = x \operatorname{arctanh} \tfrac{x}{a} + \tfrac{a}{2} \ln|a^2 - x^2|,$$

$$\textbf{38.}\ \int \operatorname{arccosh} \tfrac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \dfrac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \tfrac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \dfrac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \tfrac{x}{a} < 0 \text{ and } a > 0, \end{cases}$$

$$\textbf{39.}\ \int \frac{dx}{\sqrt{a^2 + x^2}} = \ln\left(x + \sqrt{a^2 + x^2}\right), \quad a > 0,$$

$$\textbf{40.}\ \int \frac{dx}{a^2 + x^2} = \tfrac{1}{a} \arctan \tfrac{x}{a}, \quad a > 0,$$

$$\textbf{41.}\ \int \sqrt{a^2 - x^2}\,dx = \tfrac{x}{2}\sqrt{a^2 - x^2} + \tfrac{a^2}{2} \arcsin \tfrac{x}{a}, \quad a > 0,$$

$$\textbf{42.}\ \int (a^2 - x^2)^{3/2} dx = \tfrac{x}{8}(5a^2 - 2x^2)\sqrt{a^2 - x^2} + \tfrac{3a^4}{8} \arcsin \tfrac{x}{a}, \quad a > 0,$$

$$\textbf{43.}\ \int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \tfrac{x}{a}, \quad a > 0, \qquad \textbf{44.}\ \int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln\left|\frac{a + x}{a - x}\right|, \qquad \textbf{45.}\ \int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2 - x^2}},$$

$$\textbf{46.}\ \int \sqrt{a^2 \pm x^2}\,dx = \tfrac{x}{2}\sqrt{a^2 \pm x^2} \pm \tfrac{a^2}{2} \ln\left|x + \sqrt{a^2 \pm x^2}\right|,$$

$$\textbf{47.}\ \int \frac{dx}{\sqrt{x^2 - a^2}} = \ln\left|x + \sqrt{x^2 - a^2}\right|, \quad a > 0,$$

$$\textbf{48.}\ \int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln\left|\frac{x}{a + bx}\right|,$$

$$\textbf{49.}\ \int x\sqrt{a + bx}\,dx = \frac{2(3bx - 2a)(a + bx)^{3/2}}{15b^2},$$

$$\textbf{50.}\ \int \frac{\sqrt{a + bx}}{x}\,dx = 2\sqrt{a + bx} + a \int \frac{1}{x\sqrt{a + bx}}dx,$$

$$\textbf{51.}\ \int \frac{x}{\sqrt{a + bx}}\,dx = \frac{1}{\sqrt{2}} \ln\left|\frac{\sqrt{a + bx} - \sqrt{a}}{\sqrt{a + bx} + \sqrt{a}}\right|, \quad a > 0,$$

$$\textbf{52.}\ \int \frac{\sqrt{a^2 - x^2}}{x}\,dx = \sqrt{a^2 - x^2} - a \ln\left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$$

$$\textbf{53.}\ \int x\sqrt{a^2 - x^2}\,dx = -\tfrac{1}{3}(a^2 - x^2)^{3/2},$$

$$\textbf{54.}\ \int x^2\sqrt{a^2 - x^2}\,dx = \tfrac{x}{8}(2x^2 - a^2)\sqrt{a^2 - x^2} + \tfrac{a^4}{8} \arcsin \tfrac{x}{a}, \quad a > 0,$$

$$\textbf{55.}\ \int \frac{dx}{\sqrt{a^2 - x^2}} = -\tfrac{1}{a} \ln\left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$$

$$\textbf{56.}\ \int \frac{x\,dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$$

$$\textbf{57.}\ \int \frac{x^2\,dx}{\sqrt{a^2 - x^2}} = -\tfrac{x}{2}\sqrt{a^2 - x^2} + \tfrac{a^2}{2} \arcsin \tfrac{x}{a,} \quad a > 0,$$

$$\textbf{58.}\ \int \frac{\sqrt{a^2 + x^2}}{x}\,dx = \sqrt{a^2 + x^2} - a \ln\left|\frac{a + \sqrt{a^2 + x^2}}{x}\right|,$$

$$\textbf{59.}\ \int \frac{\sqrt{x^2 - a^2}}{x}\,dx = \sqrt{x^2 - a^2} - a \arccos \tfrac{a}{|x|}, \quad a > 0,$$

$$\textbf{60.}\ \int x\sqrt{x^2 \pm a^2}\,dx = \tfrac{1}{3}(x^2 \pm a^2)^{3/2},$$

$$\textbf{61.}\ \int \frac{dx}{x\sqrt{x^2 + a^2}} = \tfrac{1}{a} \ln\left|\frac{x}{a + \sqrt{a^2 + x^2}}\right|,$$

**Theoretical Computer Science Cheat Sheet**

Calculus Cont.

**62.** $\displaystyle\int \frac{dx}{x\sqrt{x^2-a^2}} = \tfrac{1}{a}\arccos\tfrac{a}{|x|}, \quad a>0,$

**63.** $\displaystyle\int \frac{dx}{x^2\sqrt{x^2\pm a^2}} = \mp\frac{\sqrt{x^2\pm a^2}}{a^2x},$

**64.** $\displaystyle\int \frac{x\,dx}{\sqrt{x^2\pm a^2}} = \sqrt{x^2\pm a^2},$

**65.** $\displaystyle\int \frac{\sqrt{x^2\pm a^2}}{x^4}\,dx = \mp\frac{(x^2+a^2)^{3/2}}{3a^2x^3},$

**66.** $$\int \frac{dx}{ax^2+bx+c} = \begin{cases} \dfrac{1}{\sqrt{b^2-4ac}}\ln\left|\dfrac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|, & \text{if } b^2>4ac, \\ \dfrac{2}{\sqrt{4ac-b^2}}\arctan\dfrac{2ax+b}{\sqrt{4ac-b^2}}, & \text{if } b^2<4ac, \end{cases}$$

**67.** $$\int \frac{dx}{\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{1}{\sqrt{a}}\ln\left|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}\right|, & \text{if } a>0, \\ \dfrac{1}{\sqrt{-a}}\arcsin\dfrac{-2ax-b}{\sqrt{b^2-4ac}}, & \text{if } a<0, \end{cases}$$

**68.** $$\int \sqrt{ax^2+bx+c}\,dx = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ax-b^2}{8a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

**69.** $$\int \frac{x\,dx}{\sqrt{ax^2+bx+c}} = \frac{\sqrt{ax^2+bx+c}}{a} - \frac{b}{2a}\int\frac{dx}{\sqrt{ax^2+bx+c}},$$

**70.** $$\int \frac{dx}{x\sqrt{ax^2+bx+c}} = \begin{cases} \dfrac{-1}{\sqrt{c}}\ln\left|\dfrac{2\sqrt{c}\sqrt{ax^2+bx+c}+bx+2c}{x}\right|, & \text{if } c>0, \\ \dfrac{1}{\sqrt{-c}}\arcsin\dfrac{bx+2c}{|x|\sqrt{b^2-4ac}}, & \text{if } c<0, \end{cases}$$

**71.** $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac{1}{3}x^2-\tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

**72.** $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac{1}{a}x^n\cos(ax)+\tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

**73.** $\displaystyle\int x^n\cos(ax)\,dx = \tfrac{1}{a}x^n\sin(ax)-\tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

**74.** $\displaystyle\int x^ne^{ax}\,dx = \frac{x^ne^{ax}}{a}-\tfrac{n}{a}\int x^{n-1}e^{ax}\,dx,$

**75.** $\displaystyle\int x^n\ln(ax)\,dx = x^{n+1}\left(\frac{\ln(ax)}{n+1}-\frac{1}{(n+1)^2}\right),$

**76.** $\displaystyle\int x^n(\ln ax)^m\,dx = \frac{x^{n+1}}{n+1}(\ln ax)^m-\frac{m}{n+1}\int x^n(\ln ax)^{m-1}\,dx.$

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

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

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1)-f(x),$$
$$\mathrm{E}\,f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x)=\Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x)+C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

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

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

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

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

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

Sums:

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

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

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

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

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

Falling Factorial Powers:

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

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

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

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

Rising Factorial Powers:

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

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

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

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

Conversion:

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

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

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

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

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

**Theoretical Computer Science Cheat Sheet**

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \cdots &&= \sum_{i=0}^{\infty} x^i,\\
\frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \cdots &&= \sum_{i=0}^{\infty} c^i x^i,\\
\frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \cdots &&= \sum_{i=0}^{\infty} x^{ni},\\
\frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \cdots &&= \sum_{i=0}^{\infty} ix^i,\\
x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots &&= \sum_{i=0}^{\infty} i^n x^i,\\
e^x &= 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{x^i}{i!},\\
\ln(1+x) &= x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots &&= \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i},\\
\ln\frac{1}{1-x} &= x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots &&= \sum_{i=1}^{\infty} \frac{x^i}{i},\\
\sin x &= x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},\\
\cos x &= 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},\\
\tan^{-1} x &= x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},\\
(1+x)^n &= 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{n}{i} x^i,\\
\frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,\\
\frac{x}{e^x - 1} &= 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots &&= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},\\
\frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,\\
\frac{1}{\sqrt{1-4x}} &= 1 + x + 2x^2 + 6x^3 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i}{i} x^i,\\
\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n &= 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,\\
\frac{1}{1-x}\ln\frac{1}{1-x} &= x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots &&= \sum_{i=1}^{\infty} H_i x^i,\\
\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 &= \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots &&= \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i},\\
\frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \cdots &&= \sum_{i=0}^{\infty} F_i x^i,\\
\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} &= F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots &&= \sum_{i=0}^{\infty} F_{ni} x^i.
\end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\begin{aligned}
\alpha A(x) + \beta B(x) &= \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,\\
x^k A(x) &= \sum_{i=k}^{\infty} a_{i-k} x^i,\\
\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} &= \sum_{i=0}^{\infty} a_{i+k} x^i,\\
A(cx) &= \sum_{i=0}^{\infty} c^i a_i x^i,\\
A'(x) &= \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,\\
xA'(x) &= \sum_{i=1}^{\infty} i a_i x^i,\\
\int A(x)\,dx &= \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,\\
\frac{A(x) + A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i} x^{2i},\\
\frac{A(x) - A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.
\end{aligned}$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left( \sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker

## Theoretical Computer Science Cheat Sheet

### Series

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i, \qquad \left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\left\{{i \atop n}\right\}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\left[{n \atop i}\right]x^i, \qquad (e^x-1)^n = \sum_{i=0}^{\infty}\left\{{i \atop n}\right\}\frac{n!x^i}{i!},$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\left[{i \atop n}\right]\frac{n!x^i}{i!}, \qquad x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^iB_{2i}x^{2i}}{(2i)!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!}, \qquad \zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x}, \qquad \frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$

$$\zeta(x) = \prod_p\frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\textstyle\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\textstyle\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

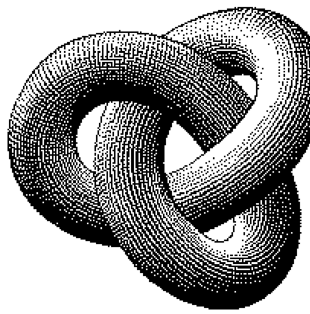
$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^ii!^2}{(i+1)(2i+1)!}x^{2i}.$$

### Escher's Knot



### Stieltjes Integration

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

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

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

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

If the integrals involved exist

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

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

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

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

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

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

### Cramer's Rule

If we have equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1\\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2\\ \vdots \qquad\quad \vdots \qquad\qquad\quad &\vdots\\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let $A=(a_{i,j})$ and $B$ be the column matrix $(b_i)$. Then there is a unique solution iff $\det A\ne 0$. Let $A_i$ be $A$ with column $i$ replaced by $B$. Then

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

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

### Fibonacci Numbers

$1,1,2,3,5,8,13,21,34,55,89,\ldots$

Definitions:

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

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

$$F_i = \frac{1}{\sqrt5}\left(\phi^i-\hat\phi^i\right),$$

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

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

Additive rule:

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

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

Calculation by matrices:

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

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

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

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

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