
Contents

1	几何	5
1.1	几何公式	5
1.2	注意	7
1.3	geo(猛犸也钻地)	7
1.4	geo	19
1.5	geo3d	24
1.6	三维几何	28
1.7	三角形	37
1.8	任意维空间最近点对	38
1.9	圆	40
1.10	圆并	42
1.11	球面	44
1.12	网格 (pick)	45
2	组合	46
2.1	组合公式	46
2.2	字典序全排列	47
2.3	字典序组合	47
2.4	排列组合生成	48
2.5	生成 gray 码	49
2.6	置换 (polya)	49
3	结构	51
3.1	ST 表	51
3.2	Splay	53
3.3	划分树	58
3.4	动态树	59
3.5	子阵和	62
3.6	左偏树	63
3.7	并查集	64
3.8	扩展并查集	65
3.9	树状数组	66
3.10	树链剖分	67
3.11	矩形并	68
3.12	线段树	70
3.13	线段树扩展	72

4	数论	75
4.1	整除规则	75
4.2	分解质因数	75
4.3	同余方程合并	78
4.4	数论变换	79
4.5	模线性方程 (组)	80
4.6	欧拉函数	81
4.7	离散对数及原根	82
4.8	简易素数表	84
4.9	阶乘最后非零位	85
4.10	高级素数表	85
5	数值	89
5.1	FFT	89
5.2	周期性方程 (追赶法)	89
5.3	多项式求根 (牛顿法)	90
5.4	定积分计算 (Romberg)	91
5.5	定积分计算 (变步长 simpson)	93
5.6	定积分计算 (自适应 simpson)	93
5.7	线性相关	94
5.8	线性规划	95
5.9	高斯消元 (全主元)	96
5.10	高斯消元 (列主元)	97
6	字符串	99
6.1	Trie 图 (dd engi)	99
6.2	Trie 图 (猛犸也钻地)	101
6.3	后缀数组 - 线性	102
6.4	后缀数组	103
6.5	后缀自动机	104
6.6	回文树	106
6.7	字符串最小表示	108
6.8	最长回文子串	109
6.9	模式匹配 (KMP+Z)	109
6.10	模式匹配 (kmp)	110
7	图论	111
7.1	NP 搜索	111
7.1.1	带权最大团	111
7.1.2	最大团 (n 小于 64)(faster)	115
7.1.3	最大团	117
7.2	匹配	118
7.2.1	一般图匹配 (Blossom)	118
7.2.2	二分图最佳匹配 (kuhn munkras 邻接阵形式)	119
7.2.3	二分图最佳匹配 (kuhn munkras 邻接阵形式)yxdb	120
7.2.4	二分图最大匹配 (hopcroft kart 邻接表形式)	121
7.2.5	二分图最大匹配 (hungary bfs 邻接阵形式)	123
7.2.6	二分图最大匹配 (hungary dfs 邻接阵形式)	123
7.3	应用	124

7.3.1	2-sat	124
7.3.2	前序表转化	125
7.3.3	拓扑排序 (邻接阵形式)	126
7.3.4	无向图全局最小割	126
7.3.5	无向图最小环	127
7.3.6	最佳边割集	128
7.3.7	最佳顶点割集	129
7.3.8	最小路径覆盖	131
7.3.9	最小边割集	132
7.3.10	最小顶点割集	133
7.3.11	树的优化算法	134
7.3.12	欧拉回路 (邻接阵形式)	136
7.4	生成树	136
7.4.1	多源最小树形图 (邻接阵形式)	136
7.4.2	最小生成树 (kruskal 邻接表形式)	138
7.4.3	最小生成树 (prim+priority queue 邻接阵形式)	139
7.4.4	最小生成树 (prim 邻接阵形式)	140
7.4.5	次最小生成树	141
7.5	网络流	142
7.5.1	最大流 (dinic d)	142
7.5.2	最高标号先流推进	145
7.5.3	网络流 (全功能)	146
7.6	连通性	150
7.6.1	支配树	150
7.6.2	无向图关键点、关键边和块	151
7.6.3	有向图强连通分量	152
8	应用	153
8.1	简单位操作	153
8.2	Joseph	154
8.3	位操作	155
8.4	布尔母函数	157
8.5	快速沃尔什哈达马变换	157
8.6	快速线性递推	159
8.7	最大子阵和	159
8.8	最长公共单调子序列	160
8.9	最长子序列	162
8.10	行列式求模	162
8.11	逆序对数	164
9	其他	165
9.1	分数	165
9.2	日期	166
9.3	矩阵	168
9.4	awt 基本应用	170

10 附录	172
10.1 应用	172
10.1.1 N 皇后构造解	172
10.1.2 大数 (整数类封装)	173
10.1.3 幻方构造	182
10.1.4 最大子串匹配	183
10.1.5 最大子段和	184
10.1.6 第 k 元素	185
10.1.7 骰子	185
10.2 算法描述	187
10.2.1 弦图与区间图	187
10.2.2 生成树	187
10.2.3 有向图最小均值环	188
11 Cheat Sheet	189
11.1 Theoretical Computer Science	189

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

$$H_a = b \sin C = c \sin B$$
$$= \sqrt{b^2 - \left(\frac{a^2+b^2-c^2}{2a}\right)^2}$$

- 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

$$a^2 + b^2 + c^2 + d^2 = D_1^2 + D_2^2 + 4M^2$$
$$S = D_1 D_2 \frac{\sin A}{2}$$

以下对圆的内接四边形, P 为半周长

$$ac + bd = D_1 D_2$$
$$S = \sqrt{(P-a)(P-b)(P-c)(P-d)}$$

- 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

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

- 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

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

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

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

- 球

- 全面积

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

<ul style="list-style-type: none"> - 体积 $V = \frac{4\pi r^3}{3}$	<ul style="list-style-type: none"> • 球扇形 <p>h 为球冠高, r_0 为球冠底面半径</p>
<ul style="list-style-type: none"> • 球台 	
<ul style="list-style-type: none"> - 侧面积 $S = 2\pi r h$	<ul style="list-style-type: none"> - 全面积 $T = \pi r(2h + r_0)$
<ul style="list-style-type: none"> - 全面积 $T = \pi(2rh + r_1^2 + r_2^2)$	<ul style="list-style-type: none"> - 体积 $V = \frac{2\pi r^2 h}{3}$
<ul style="list-style-type: none"> - 体积 $V = \frac{\pi h(3(r_1^2 + r_2^2) + h^2)}{6}$	

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 2π 域内两角度差 (beta) 应该是
 $\text{abs}(a1-a2) < \text{beta} \ || \ \text{abs}(a1-a2) > \text{pi} + \text{pi} - \text{beta}$
 判断相等时将 beta 换成 eps
7. 需要的话尽量使用 atan2, 注意:
 - $\text{atan2}(0,0) = 0$
 - $\text{atan2}(1,0) = \text{pi}/2$
 - $\text{atan2}(-1,0) = -\text{pi}/2$
 - $\text{atan2}(0,1) = 0$
 - $\text{atan2}(0,-1) = \text{pi}$
8. cross product = $|u||v| \sin \alpha$
 dot product = $|u||v| \cos \alpha$
9. $(P_1 - P_0) \times (P_2 - P_0)$ 结果的意义:
 - 正: $\langle P_0, P_1 \rangle$ 在 $\langle P_0, P_2 \rangle$ 顺时针 $(0, \pi)$ 内
 - 负: $\langle P_0, P_1 \rangle$ 在 $\langle P_0, P_2 \rangle$ 逆时针 $(0, \pi)$ 内
 - 0: $\langle P_0, P_1 \rangle, \langle P_0, P_2 \rangle$ 共线, 夹角为 0 或 π

1.3 geo(猛犸也钻地)

```

1 // 计算几何 By 猛犸也钻地 @ 2012.08.21
2
3 /* 命名约定 */
4 圆: 圆心在 u, 一般情况下半径 r 大于等于 0
5 直线: 经过点 u 和 v 的直线, u 不重合于 v
6 射线: 起点在 u, 途经点 v, u 不重合于 v
7 线段: 起点在 u, 终点在 v, u 不重合于 v
8 散点集: 点的可空集合
9 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
10 凸多边形: 各内角均小于 180 度的多边形
11 平面: 由不共线的三点 uvw 所表示

```

```

12 // 所有函数都会默认传入的参数已满足上面的命名约定 */
13
14 #include <vector>
15 #include <cmath>
16 #include <utility>
17 #include <algorithm>
18 using namespace std;
19 using namespace rel_ops;
20
21 // typedef Long Long NUM;
22 typedef double NUM;
23 const NUM EPS = 1e-12, MAGIC = 2.71828e18;
24 // 因为有相对误差判断, 所以 EPS 不要设得太宽
25
26 inline NUM sqr(NUM a) {
27     return a*a;
28 }
29 inline NUM cmp(NUM a, NUM b) {
30     // return a-b; // 坐标为浮点数时, 使用下面这行
31     return fabs(a-b)>=EPS+fabs(a)*EPS?a-b:0;
32 }
33
34 //-----//
35
36 struct VEC {
37     NUM x,y;
38 } NOVEC = {MAGIC,MAGIC};
39 struct RAY {
40     VEC u,v;
41 } NORAY = {NOVEC,NOVEC};
42 struct CIR {
43     VEC u;
44     NUM r;
45 } NOCIR = {NOVEC,MAGIC};
46
47 inline NUM sqr(const VEC &a) {
48     return sqr(a.x)+sqr(a.y);
49 }
50 inline double abs(const VEC &a) {
51     return sqrt(sqr(a));
52 }
53 inline NUM cmp(const VEC &a, const VEC &b) {
54     NUM at=cmp(a.x,b.x);
55     return !at?cmp(a.y,b.y):at;
56 }
57
58 inline VEC operator +(const VEC &a, const VEC &b) {
59     return (VEC) {
60         a.x+b.x,a.y+b.y
61     };
62 }
63 inline VEC operator -(const VEC &a, const VEC &b) {
64     return (VEC) {
65         a.x-b.x,a.y-b.y
66     };

```



```

67 }
68 inline NUM operator *(const VEC &a, const VEC &b) {
69     return a.x*b.y-a.y*b.x;
70 }
71 inline NUM operator %(const VEC &a, const VEC &b) {
72     return a.x*b.x+a.y*b.y;
73 }
74 inline VEC operator -(const VEC &a) {
75     return (VEC) {
76         -a.x,-a.y
77     };
78 }
79 inline VEC operator ~(const VEC &a) {
80     return (VEC) {
81         -a.y,+a.x
82     };
83 }
84 inline VEC operator *(NUM u, const VEC &a) {
85     return (VEC) {
86         u *a.x,u *a.y
87     };
88 }
89 inline VEC operator *(const VEC &a, NUM u) {
90     return (VEC) {
91         a.x *u,a.y *u
92     };
93 }
94 inline VEC operator /(const VEC &a, NUM u) {
95     return (VEC) {
96         a.x/u,a.y/u
97     };
98 }
99 inline VEC operator /(const VEC &a, const VEC &b) {
100     return a%b/sqr(b)*b;
101 }
102 inline bool operator ==(const VEC &a, const VEC &b) {
103     return !cmp(a,b);
104 }
105 inline bool operator <(const VEC &a, const VEC &b) {
106     return cmp(a,b)<0;
107 }
108
109 // 返回值      cmp_side      cmp_axis
110 // == 0      a 和 b 相互平行    /    a 和 b 相互垂直
111 // <= -EPS   a 在 b 的左手侧    /    a 和 b 朝向相反 (内角大于 90 度)
112 // >= +EPS   a 在 b 的右手侧    /    a 和 b 朝向相同 (内角小于 90 度)
113 NUM cmp_side(const VEC &a, const VEC &b) {
114     return cmp(a.x*b.y,+a.y*b.x);
115 }
116 NUM cmp_axis(const VEC &a, const VEC &b) {
117     return cmp(a.x*b.x,-a.y*b.y);
118 }
119
120 //-----//
121

```

```

122 // 求向量 a 长度缩放至 u 单位后的新向量, a 不能是零向量
123 // 求向量 a 绕坐标原点 o, 逆时针转 u 度后的新向量
124 VEC resize(const VEC &a, NUM u) {
125     return u/abs(a)*a;
126 }
127 VEC rotate(const VEC &a, NUM u) {
128     return (VEC) {
129         cos(u)*a.x-sin(u)*a.y, sin(u)*a.x+cos(u)*a.y
130     };
131 }
132
133 // 点在直线上的投影 (到直线的最近点)
134 // 点在圆周上的投影 (到圆周的最近点)
135 VEC project(const VEC &p, const RAY &l) {
136     return (p-l.u)/(l.v-l.u)+l.u;
137 }
138 VEC project(const VEC &p, const CIR &c) {
139     if(!cmp(p,c.u)) return NOVEC;
140     return resize(p-c.u,c.r)+c.u;
141 }
142
143 // 求两直线的交点
144 // 求直线与圆的交点, 交线段的方向与原先直线相同
145 // 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
146 // 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)
147 VEC intersect(const RAY &a, const RAY &b) {
148     VEC s=a.u-a.v,t=b.u-b.v;
149     NUM at=cmp_side(s,t);
150     if(!at) return NOVEC;
151     return a.u+(b.u-a.u)*t/at*s;
152 }
153 RAY intersect(const RAY &l, const CIR &c) {
154     VEC s=l.u+(c.u-l.u)/(l.v-l.u);
155     NUM at=cmp(c.r*c.r,sqr(s-c.u));
156     if(at<0) return NORAY;
157     VEC t=resize(l.v-l.u,sqrt(at));
158     return (RAY) {
159         s-t,s+t
160     };
161 }
162 RAY intersect(const CIR &a, const CIR &b) {
163     NUM l=sqr(b.u-a.u);
164     NUM w=(1+(a.r*a.r-b.r*b.r)/l)*0.5;
165     NUM e=cmp(a.r*a.r/l,w*w);
166     if(e<0) return NORAY;
167     VEC t=sqrt(e)*~(b.u-a.u);
168     VEC s=a.u+w*(b.u-a.u);
169     return (RAY) {
170         s-t,s+t
171     };
172 }
173
174 // 判断三点是否共线

```

```

175 // 判断点在直线上的投影点, 是否在线段上
176 // 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)
177 // 判断点和圆的位置关系, 在外侧为 0, 内部为 1, 边上为 2
178 // 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
179 // 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
180 // 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
181 bool collinear(const VEC &a, const VEC &b, const VEC &c) {
182     return !cmp_side(a-b,b-c);
183 }
184 bool seg_range(const VEC &p, const RAY &l) {
185     return cmp_axis(p-l.u,p-l.v)<=0;
186 }
187 int relation(const VEC &p, const RAY &l) {
188     if(cmp_side(p-l.u,p-l.v)) return 0;
189     return cmp_axis(p-l.u,p-l.v)>0?1:2;
190 }
191 int relation(const VEC &p, const CIR &c) {
192     NUM at=cmp(sqr(c.r),sqr(c.u-p));
193     return at?at<0?0:1:2;
194 }
195 int relation(const VEC &p, const vector<VEC> &u) {
196     int n=u.size(),ret=0;
197     for(int i=0; i<n; i++) {
198         VEC s=u[i]-p,t=u[(i+1)%n]-p;
199         if(t<s) swap(s,t);
200         if(!cmp_side(s,t) && cmp_axis(s,t)<=0) return 2;
201         if(cmp(s.x+p.x,p.x)<=0 && cmp(t.x+p.x,p.x)>0
202            && cmp_side(s,t)>0) ret^=1;
203     }
204     return ret;
205 }
206 int relation_convex(const VEC &p, const vector<VEC> &u) {
207     int n=u.size(),l=0,r=n-1,o=cmp_side(u[1]-u[0],u[r]-u[0])<0?-1:1;
208     if(relation(p,(RAY) {
209         u[0],u[1]
210     })==2
211        || relation(p,(RAY) {
212         u[0],u[r]
213     })==2) return 2;
214     while(l<r) {
215         int m=(l+r+1)/2;
216         if(cmp_side(p-u[0],u[m]-u[0])*o<=0) l=m;
217         else r=m-1;
218     }
219     if(!r || r==n-1) return 0;
220     NUM at=cmp_side(p-u[r],u[r+1]-u[r])*o;
221     return at?at<0:2;
222 }
223 int relation(const RAY &a, const RAY &b) {
224     NUM at=cmp_side(a.u-a.v,b.u-b.v);
225     return at?!cmp_axis(a.u-a.v,b.u-b.v):!cmp_side(a.u-b.u,a.u-b.v)+2;
226 }
227
228 // 由  $ax+by+c=0$  构造直线

```

```

229 // 由直径上的两点构造一个圆
230 // 由三角形的顶点构造外接圆
231 RAY make_line(NUM a, NUM b, NUM c) {
232     if(!cmp(a,0) && !cmp(b,0)) return NORAY;
233     else if(!cmp(a,0)) return (RAY) {
234         {0, -c/b}, {1, -c/b}
235     };
236     else if(!cmp(b,0)) return (RAY) {
237         {-c/a, 0}, {-c/a, 1}
238     };
239     return (RAY) {
240         {0, -c/b}, {-c/a, 0}
241     };
242 }
243 CIR make_circle(const VEC &a, const VEC &b) {
244     return (CIR) {
245         (a+b)/2, abs(a-b)/2
246     };
247 }
248 CIR make_circle(const VEC &a, const VEC &b, const VEC &c) {
249     if(!cmp_side(a-b, a-c)) return NOCIR;
250     NUM x=(c-b)%(a-c), y=(c-b)*(a-b);
251     VEC m=(x/y*~(a-b)+a+b)/2;
252     return (CIR) {
253         m, abs(a-m)
254     };
255 }
256
257 // 求三点的内切圆
258 // 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
259 // 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
260 // 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
261 CIR tangent_circle(const VEC &a, const VEC &b, const VEC &c) {
262     if(!cmp_side(a-b, a-c)) return NOCIR;
263     NUM x=abs(b-c), y=abs(c-a), z=abs(a-b);
264     VEC m=(a*x+b*y+c*z)/(x+y+z);
265     return (CIR) {
266         m, fabs((m-a)*(a-b)*1.0/z)
267     };
268 }
269 RAY tangent(const VEC &p, const CIR &c) {
270     NUM l=sqr(p-c.u), e=cmp(1, c.r*c.r);
271     if(e<0) return NORAY;
272     NUM x=c.r/sqrt(l), y=sqrt(e/l);
273     VEC s=resize(p-c.u, 1), t=~s;
274     RAY lr= {c.u+c.r *x *s-c.r *y*t,
275             c.u+c.r *x *s+c.r *y*t
276     };
277     return lr;
278 }
279 pair<RAY, RAY> tangent(const CIR &a, const CIR &b) {
280     NUM o=a.r-b.r, l=sqr(b.u-a.u), e=cmp(1, o*o);
281     if(e<0) return make_pair(NORAY, NORAY);
282     NUM x=o/sqrt(l), y=sqrt(e/l);
283     VEC s=resize(b.u-a.u, 1), t=~s;

```

```

284   RAY ll= {a.u+a.r *x *s+a.r *y*t,
285           b.u+b.r *x *s+b.r *y*t
286           };
287   RAY rr= {a.u+a.r *x *s-a.r *y*t,
288           b.u+b.r *x *s-b.r *y*t
289           };
290   return make_pair(ll,rr);
291 }
292
293 // 由散点集构造一个最小覆盖圆, 期望复杂度  $O(n)$ 
294 CIR min_covering_circle(vector<VEC> u) {
295     random_shuffle(u.begin(),u.end());
296     int n=u.size(),i,j,k,z=1%n;
297     CIR ret;
298     for(ret=make_circle(u[0],u[z]),i=2; i<n; i++) if(!relation(u[i],ret))
299         for(ret=make_circle(u[0],u[i]),j=1; j<i; j++) if(!relation(u[j],ret))
300             for(ret=make_circle(u[i],u[j]),k=0; k<j; k++) if(!relation(u[k],ret)\
301 )
302         ret=make_circle(u[i],u[j],u[k]);
303     return ret;
304 }
305
306 // 求散点集的二维凸包, 并按逆时针顺序排列
307 // 若传入的点集不足以构成凸多边形, 则返回的点集是退化后的点或线段
308 vector<VEC> convex_hull(vector<VEC> u) {
309     sort(u.begin(),u.end()); // 这两行是排序 + 去重, 如果数据已经有保证
310     u.erase(unique(u.begin(),u.end()),u.end()); // 则可省略相应的操作
311     if(u.size()<3) return u;
312     vector<VEC> c;
313     for(size_t i=0,o=1,m=1; ~i; i+=o) {
314         while(c.size()>m) {
315             VEC a=c.back()-c[c.size()-2];
316             VEC b=c.back()-u[i];
317             if(cmp_side(a,b)<0) break; // 改成 <=0 则保留共线点
318             c.pop_back();
319         }
320         c.push_back(u[i]);
321         if(i+1==u.size()) m=c.size(),o=-1; // 条件成立时切换至上凸壳
322     }
323     c.pop_back();
324     return c;
325 }
326
327 /* 警告: 下面这两个函数没有被正确地实现, 等待修正中
328     比赛时请不要使用这两个函数, 有较高几率出错
329
330 // 求凸多边形上, 朝某个方向看过去的最远点的编号, 复杂度  $O(\log n)$ 
331 // 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
332 int apoapsis(const VEC& v, const vector<VEC>& u){
333     if(!cmp((VEC){0,0},v)) return -1;

```

```

334     int l=0,r=u.size()-1;
335     NUM s=cmp_axis(u[r]-u[0],v);
336     NUM t=cmp_axis(u[1]-u[0],v);
337     if(s<=0 && t<=0) return !s?r:0;
338     while(l<r){
339         int m=(l+r)/2,e=cmp_axis(u[m]-u[0],v);
340         if((e>=0 && e<cmp_axis(u[m+1]-u[0],v))
341            || (e<0 && t<0)) l=m+1; else r=m;
342     }
343     return r;
344 }
345
346 // 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  $O(\log n)$ 
347 RAY intersect(RAY L, const vector<VEC>& u){
348     int n=u.size(),p,q,lo,hi;
349     VEC o=L.v-L.u;
350     if(cmp_side(u[1]-u[0],u[2]-u[0])<0) o=-o;
351     NUM pt=cmp_side(o,u[p=apoapsis(~ o,u)]-L.u);
352     NUM qt=cmp_side(o,u[q=apoapsis(~-o,u)]-L.u);
353     if(pt*qt>0) return NORAY;
354     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
355         lo=p,hi=q+=n;
356         swap(p+=n,q);
357         while(lo<hi){
358             int at=(lo+hi+1)/2;
359             if(cmp_side(o,u[at%n]-L.u)>=0) lo=at; else hi=at-1;
360         }
361         if(!cmp_side(o,u[lo%n]-L.u)) L.u=u[lo%n];
362         else L.u=intersect((RAY){u[lo%n],u[(lo+1)%n]},L);
363         swap(L.u,L.v);
364     }
365     return L;
366 }
367
368 /*-----*/
369
370 struct TOR {
371     NUM x,y,z;
372 } NOTOR = {MAGIC,MAGIC,MAGIC};

```

```

373 struct SIG {
374     TOR u,v;
375 } NOSIG = {NOTOR,NOTOR};
376 struct PLN {
377     TOR u,v,w;
378 } NOPLN = {NOTOR,NOTOR,NOTOR};
379
380 inline NUM sqr(const TOR &a) {
381     return sqr(a.x)+sqr(a.y)+sqr(a.z);
382 }
383 inline double abs(const TOR &a) {
384     return sqrt(sqr(a));
385 }
386 inline NUM cmp(const TOR &a, const TOR &b) {
387     NUM at=cmp(a.x,b.x);
388     if(!at) at=cmp(a.y,b.y);
389     return !at?cmp(a.z,b.z):at;
390 }
391
392 inline TOR operator +(const TOR &a, const TOR &b) {
393     return (TOR) {
394         a.x+b.x,a.y+b.y,a.z+b.z
395     };
396 }
397 inline TOR operator -(const TOR &a, const TOR &b) {
398     return (TOR) {
399         a.x-b.x,a.y-b.y,a.z-b.z
400     };
401 }
402 inline TOR operator *(const TOR &a, const TOR &b) {
403     return (TOR) {
404         a.y *b.z-a.z *b.y,a.z *b.x-a.x *b.z,a.x *b.y-a.y *b.x
405     };
406 }
407 inline NUM operator %(const TOR &a, const TOR &b) {
408     return a.x*b.x+a.y*b.y+a.z*b.z;
409 }
410 inline TOR operator -(const TOR &a) {
411     return (TOR) {
412         -a.x,-a.y,-a.z
413     };
414 }
415 inline TOR operator *(NUM u, const TOR &a) {
416     return (TOR) {
417         u *a.x,u *a.y,u *a.z
418     };
419 }
420 inline TOR operator *(const TOR &a, NUM u) {
421     return (TOR) {
422         a.x *u,a.y *u,a.z *u
423     };
424 }
425 inline TOR operator /(const TOR &a, NUM u) {
426     return (TOR) {
427         a.x/u,a.y/u,a.z/u
428     };
429 }
430 inline TOR operator /(const TOR &a, const TOR &b) {

```

```

431     return a%b/sqr(b)*b;
432 }
433 inline bool operator ==(const TOR &a, const TOR &b) {
434     return !cmp(a,b);
435 }
436 inline bool operator <(const TOR &a, const TOR &b) {
437     return cmp(a,b)<0;
438 }
439
440 // 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
441 int cmp_side(const TOR &a, const TOR &b) {
442     return cmp(a.y*b.z,a.z*b.y)
443         || cmp(a.z*b.x,a.x*b.z)
444         || cmp(a.x*b.y,a.y*b.x);
445 }
446 NUM cmp_axis(const TOR &a, const TOR &b) {
447     NUM x=a.x*b.x,y=a.y*b.y,z=a.z*b.z;
448     if((x<0)==(y<0)) return cmp(x+y,-z);
449     if((x<0)==(z<0)) return cmp(x+z,-y);
450     // 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
451     return cmp(y+z,-x);
452 }
453
454 //-----//
455
456 // 求平面 c 的法向量
457 // 求向量 a 长度缩放至 u 单位后的新向量, a 不能是零向量
458 // 求向量 a 绕转向量 o, 逆时针转 u 度后的新向量
459 inline TOR normal(const PLN &c) {
460     return (c.v-c.u)*(c.w-c.u);
461 }
462 TOR resize(const TOR &a, NUM u) {
463     return u/abs(a)*a;
464 }
465 TOR rotate(const TOR &a, NUM u, const TOR &o) {
466     return a*cos(u)+resize(o,1)*a*sin(u);
467 }
468
469 // 点在直线上的投影 (到直线的最近点)
470 // 点在平面上的投影 (到平面的最近点)
471 TOR project(const TOR &p, const SIG &l) {
472     return (p-l.u)/(l.v-l.u)+l.u;
473 }
474 TOR project(const TOR &p, const PLN &c) {
475     return (c.u-p)/normal(c)+p;
476 }
477
478 // 求两直线的交点
479 // 求直线与平面的交点
480 // 求两平面的交线
481 TOR intersect(const SIG &a, const SIG &b) {

```



```

482     TOR s=b.u-b.v,p=s*(b.u-a.u);
483     TOR t=a.u-a.v,q=s*t;
484     if(cmp_axis(p,t) || !cmp_side(s,t)) return NOTOR;
485     NUM at=cmp_axis(p,q);
486     return a.u+(at?at<0?-1:1:0)*sqrt(sqr(p)/sqr(q))*t;
487 }
488 TOR intersect(const SIG &l, const PLN &c) {
489     TOR at=l.v-l.u,o=normal(c);
490     if(!cmp_axis(o,at)) return NOTOR;
491     return l.u+(c.u-l.u)%o/(at%o)*at;
492 }
493 SIG intersect(const PLN &a, const PLN &b) {
494     TOR o=normal(a);
495     SIG s= {b.u,b.v},t= {b.u,b.w},r= {b.v,b.w};
496     s.u=intersect(cmp_axis(s.u-s.v,o)?s:r,a);
497     t.u=intersect(cmp_axis(t.u-t.v,o)?t:r,a);
498     return (SIG) {
499         s.u,t.u
500     };
501 }
502
503 // 判断四点是否共面
504 // 判断三点是否共线
505 // 判断点在直线上的投影点, 是否在线段上
506 // 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)
507 // 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
508 // 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
509 // 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
510 // 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
511 bool coplanar(const TOR &a, const TOR &b, const TOR &c, const TOR &d) {
512     return !cmp_axis(a-b,(a-c)*(a-d));
513 }
514 bool collinear(const TOR &a, const TOR &b, const TOR &c) {
515     return !cmp_side(a-b,b-c);
516 }
517 bool seg_range(const TOR &p, const SIG &l) {
518     return cmp_axis(p-l.u,p-l.v)<=0;
519 }
520 int relation(const TOR &p, const SIG &l) {
521     if(cmp_side(p-l.u,p-l.v)) return 0;
522     return cmp_axis(p-l.u,p-l.v)>0?1:2;
523 }
524 int relation(const TOR &p, const PLN &c) {
525     NUM at=cmp_axis(p-c.u,normal(c));
526     return at?at<0?-1:1:0;
527 }
528 int relation(const SIG &a, const SIG &b) { // 注意, 异面垂直也算垂直
529     NUM at=cmp_side(a.u-a.v,b.u-b.v);
530     return at?!cmp_axis(a.u-a.v,b.u-b.v):!cmp_side(a.u-b.u,a.u-b.v)+2;
531 }
532 int relation(const SIG &l, const PLN &c) {
533     TOR o=normal(c),e=l.v-l.u;

```

```

534     return cmp_axis(e,o)?!cmp_side(e,o):!cmp_axis(c.u-1.u,o)+2;
535 }
536 int relation(const PLN &a, const PLN &b) {
537     TOR p=normal(a),q=normal(b);
538     return cmp_side(p,q)?!cmp_axis(p,q):!cmp_axis(a.u-b.u,p)+2;
539 }
540
541 // 由  $ax+by+cz+d=0$  构造平面
542 PLN make_plane(NUM a, NUM b, NUM c, NUM d) {
543     if(cmp(a,0)) return (PLN) {
544         {-d/a,0,0}, {(-b-d)/a,1,0}, {(-c-d)/a,0,1}
545     };
546     if(cmp(b,0)) return (PLN) {
547         {0,-d/b,0}, {1,(-a-d)/b,0}, {0,(-c-d)/b,1}
548     };
549     if(cmp(c,0)) return (PLN) {
550         {0,0,-d/c}, {1,0,(-a-d)/c}, {0,1,(-b-d)/c}
551     };
552     return NOPLN;
553 }
554
555 // 求散点集的三维凸包, 返回每个三角面的顶点编号, 复杂度  $O(n\log n)$ 
556 // 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
557 // 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
558 // 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
559 // 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
560 struct TPL {
561     int u[3];
562 };
563 vector<TPL> convex_hull(const vector<TOR> &p) {
564     vector<TPL> face, edge;
565     static vector<int> F[100005], G[100005*7]; // 注意设置最大结点数
566     int n=p.size(), i, j, k;
567     if(n<=3) return face;
568     vector<int> u(n), v(4), at(n), go(n), by(n);
569     for(i=0; i<n; i++) u[i]=i;
570     random_shuffle(u.begin(), u.end());
571     TOR a=p[u[0]]-p[u[1]], b;
572     for(i=2; i<n; i++) if(cmp_side(a, b=p[u[0]]-p[u[i]])) break;
573     for(j=i; j<n; j++) if(cmp_axis(a*b, p[u[0]]-p[u[j]])) break;
574     if(i>=n || j>=n) return face;
575     swap(u[i], u[2]), swap(u[j], u[3]);
576     b=p[u[0]]+p[u[1]]+p[u[2]]+p[u[3]];
577     for(i=0; i<4; i++) {
578         a=(p[u[i]]-p[u[j=(i+1)%4]])*(p[u[i]]-p[u[k=(i+2)%4]]);
579         if(cmp_axis(p[u[i]]*4-b, a)<0) swap(j, k), a=-a;
580         face.push_back((TPL) {
581             {
582                 u[i], u[j], u[k]
583             }
584         });
585         edge.push_back((TPL) {
586             {

```

```

587         (k+1)%4,(i+1)%4,(j+1)%4
588     }
589 });
590     for(j=4; j<n; j++) if(cmp_axis(p[u[j]]-p[u[i]],a)>0)
591         F[j].push_back(i),G[i].push_back(j);
592 }
593     for(i=4; i<n; F[i++].clear()) {
594         int x=n,m=F[i].size(),c=v.size();
595         for(j=0; j<m; j++) v[F[i][j]]++;
596         for(j=0; j<m; j++) if(v[F[i][j]]>0) {
597             v[F[i][j]]=-1234567890;
598             for(k=0; k<3; k++) {
599                 if(v[edge[F[i][j]].u[k]]) continue;
600                 at[x=face[F[i][j]].u[k]]=F[i][j];
601                 go[x]=k;
602             }
603         }
604         if(x==n) continue;
605         for(j=x,k=-1; k!=x; j=k) {
606             k=face[at[j]].u[(go[j]+1)%3];
607             a=(p[j]-p[k])*(p[j]-p[u[i]]);
608             int t=v.size(),w=edge[at[j]].u[go[j]];
609             v.push_back(0);
610             face.push_back((TPL) {
611                 {
612                     j,k,u[i]
613                 }
614             });
615             edge.push_back((TPL) {
616                 {
617                     w,t+1,t-1
618                 }
619             });
620             *find(edge[w].u,edge[w].u+3,at[j])=t;
621             vector<int>::const_iterator o,z;
622             z=set_union(G[at[j]].begin(),G[at[j]].end(),
623                 G[w].begin(),G[w].end(),by.begin());
624             for(o=by.begin(); o!=z; ++o)
625                 if(*o>i && cmp_axis(p[u[*o]]-p[u[i]],a)>0)
626                     F[*o].push_back(t),G[t].push_back(*o);
627         }
628         edge[edge.back().u[1]=c].u[2]=edge.size()-1;
629     }
630     int m=v.size();
631     for(i=j=0; i<m; G[i++].clear())
632         if(!v[i]) face[j]=face[i],edge[j++]=edge[i];
633     face.erase(face.begin()+j,face.end());
634     edge.erase(edge.begin()+j,edge.end());
635     return face;
636 }

```

1.4 geo

```

1 #include <cmath>
2 #include <algorithm>
3 using namespace std;

```

```

4  const int MAXN = 1000;
5  const double eps = 1e-8, PI = atan2(0, -1);
6  inline double sqr(double x) {
7      return x * x;
8  }
9  inline bool zero(double x) {
10     return (x > 0 ? x : -x) < eps;
11 }
12 inline int sgn(double x) {
13     return (x > eps ? 1 : (x + eps < 0 ? -1 : 0));
14 }
15 struct point {
16     double x, y;
17     point(double x, double y):x(x), y(y) {}
18     point() {}
19     bool operator == (const point &a) const {
20         return sgn(x - a.x) == 0 && sgn(y - a.y) == 0;
21     }
22     bool operator != (const point &a) const {
23         return sgn(x - a.x) != 0 || sgn(y - a.y) != 0;
24     }
25     bool operator < (const point &a) const {
26         return sgn(x - a.x) < 0 || sgn(x - a.x) == 0 && sgn(y - a.y) < 0;
27     }
28     point operator + (const point &a) const {
29         return point(x + a.x, y + a.y);
30     }
31     point operator - (const point &a) const {
32         return point(x - a.x, y - a.y);
33     }
34     point operator * (const double &a) const {
35         return point(x * a, y * a);
36     }
37     point operator / (const double &a) const {
38         return point(x / a, y / a);
39     }
40     double operator * (const point &a) const {
41         return x * a.y - y * a.x; //xmult
42     }
43     double operator ^ (const point &a) const {
44         return x * a.x + y * a.y; //dmult
45     }
46     double length() const {
47         return sqrt(sqr(x) + sqr(y));
48     }
49     point trunc(double a) const {
50         return (*this) * (a / length());
51     }
52     point rotate(double ang) const {
53         point p(sin(ang), cos(ang));
54         return point((*this) * p, (*this) ^ p);
55     }
56     point rotate(const point &a) const {
57         point p(-a.y, a.x);
58         p = p.trunc(1.0);
59         return point((*this) * p, (*this) ^ p);
60     }

```

```

61 };
62 bool isConvex(int n, const point *p) {
63     int i, s[3] = {1, 1, 1};
64     for(i = 0; i < n && /*s[1] && */ s[0] | s[2]; i++)
65         s[sgn((p[(i + 1) % n] - p[i]) * (p[(i + 2) % n] - p[i])) + 1] = 0;
66     return /*s[1] && */ s[0] | s[2];
67 } //去掉注释即不允许相邻边共线
68 bool insideConvex(const point &q, int n, const point *p) {
69     int i, s[3] = {1, 1, 1};
70     for(i = 0; i < n && /*s[1] && */ s[0] | s[2]; i++)
71         s[sgn((p[(i + 1) % n] - p[i]) * (q - p[i])) + 1] = 0;
72     return /*s[1] && */ s[0] | s[2];
73 } //去掉注释即严格在形内
74 inline bool dotsInline(const point &p1, const point &p2, const point &p3) {
75     return zero((p1 - p3) * (p2 - p3));
76 } //三点共线
77 inline int decideSide(const point &p1, const point &p2, const point &l1, const point &l2\
78 ) {
79     return sgn((l1 - l2) * (p1 - l2)) * sgn((l1 - l2) * (p2 - l2));
80 } //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
81 inline bool dotOnlineIn(const point &p, const point &l1, const point &l2) {
82     return zero((p - l2) * (l1 - l2)) && (l1.x - p.x) * (l2.x - p.x) < eps && (l1.y - p.\
83 y) * (l2.y - p.y) < eps;
84 } //判点是否在线段及其端点上
85 inline bool parallel(const point &u1, const point &u2, const point &v1, const point &v2)\
86 {
87     return zero((u1 - u2) * (v1 - v2));
88 } //判直线平行
89 inline bool perpendicular(const point &u1, const point &u2, const point &v1, const point\
90 &v2) {
91     return zero((u1 - u2) ^ (v1 - v2));
92 } //判直线垂直
93 inline bool intersectIn(const point &u1, const point &u2, const point &v1, const point &\
94 v2) {
95     if(!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2))
96         return decideSide(u1, u2, v1, v2) != 1 && decideSide(v1, v2, u1, u2) != 1;
97     else
98         return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1, u1,\
99 u2) || dotOnlineIn(v2, u1, u2);
100 } //判两线段相交, 包括端点和部分重合
101 inline bool intersectEx(const point &u1, const point &u2, const point &v1, const point &\
102 v2) {
103     return decideSide(u1, u2, v1, v2) < 0 && decideSide(v1, v2, u1, u2) < 0;
104 } //判两线段相交, 不包括端点和部分重合
105 inline bool insidePolygon(const point &q, int n, const point *p, bool onEdge = true) {
106     if(dotOnlineIn(q, p[n - 1], p[0])) return onEdge;
107     for(int i = 0; i + 1 < n; i++) if(dotOnlineIn(q, p[i], p[i + 1])) return onEdge;
108 #define getq(i) Q[(sgn(p[i].x-q.x)>0)<<1|sgn(p[i].y-q.y)>0]
109 #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

```

```

110 1))))
111     int Q[4] = {2, 1, 3, 0}, oq = getq(n-1), nq = getq(0), qua = difq(nq, oq, n - 1, 0);
112     oq = nq;
113     for(int i = 1; i < n; i++) {
114         nq = getq(i);
115         qua += difq(nq, oq, i - 1, i);
116         oq = nq;
117     }
118     return qua != 0; //象限环顾法, 较好
119     /*point q1; int i = 0, cnt = 0; const double OFFSET = 1e6; //坐标上限
120     for(q1 = point(rand() + OFFSET, rand() + OFFSET); i < n; i++) for(i = cnt = 0; i < n && !d\
121     otsInline(q, q1, p[i]); i++) cnt += intersectEx(q, q1, p[i], p[(i + 1) % n]);
122     return cnt & 1; */ //考验 rp 的射线法
123 } //判点在任意多边形内
124 inline point intersection(const point &u1, const point &u2, const point &v1, const point &v2) {
125     return u1 + (u2 - u1) * (((u1 - v1) * (v1 - v2)) / ((u1 - u2) * (v1 - v2)));
126 } //求两直线交点, 须预判是否平行
127 inline point ptoline(const point &p, const point &l1, const point &l2) {
128     point t = p;
129     t.x += l1.y - l2.y;
130     t.y += l2.x - l1.x;
131     return intersection(p, t, l1, l2);
132 } //点到直线的最近点, 注意 l1 不能等于 l2
133 inline double disptoline(const point &p, const point &l1, const point &l2) {
134     return fabs((p - l2) * (l1 - l2)) / (l1 - l2).length();
135 } //点到直线距离, 注意 l1 不能等于 l2
136 inline point ptoseg(const point &p, const point &l1, const point &l2) {
137     point t = p;
138     t.x += l1.y - l2.y;
139     t.y += l2.x - l1.x;
140     if(sgn((l1 - p) * (t - p)) * sgn((l2 - p) * (t - p)) > 0)
141         return (p - l1).length() < (p - l2).length() ? l1 : l2;
142     else
143         return intersection(p, t, l1, l2);
144 } //点到线段的最近点, 注意 l1 不能等于 l2
145 inline double disptoseg(const point &p, const point &l1, const point &l2) {
146     point t = point(l1.y - l2.y, l2.x - l1.x);
147     if(sgn((l1 - p) * t) * sgn((l2 - p) * t) > 0)
148         return min((p - l1).length(), (p - l2).length());
149     else
150         return disptoline(p, l1, l2);
151 } //点到线段距离, 注意 l1 不能等于 l2
152 double fermentpoint(int m, point p[]) {
153     point u(0, 0), v;
154     double step = 0, nowbest = 0, now, maxx = 0, maxy = 0;
155     for(int i = 0; i < m; ++i) {
156         u = u + p[i];
157         maxx = max(maxx, fabs(p[i].x));
158         maxy = max(maxy, fabs(p[i].y));
159     }
160 }

```

```

161     u = u / m;
162     for(int i = 0; i < m; ++i) nowbest += (u - p[i]).length();
163     for(step = maxx + maxy; step > 1e-10; step *= 0.97) //对结果有影响, 注意调整
164         for(int i = -1; i <= 1; i++)
165             for(int j = -1; j <= 1; j++) {
166                 v = u + point(i, j) * step;
167                 now = 0;
168                 for(int i = 0; i < m; ++i) now += (v - p[i]).length();
169                 if(now < nowbest) {
170                     nowbest = now;
171                     u = v;
172                 }
173             }
174     return nowbest;
175 } //模拟退火求费马点
176 void polygonCut(int &n, point *p, const point &l1, const point &l2, const point &side) {
177     int m = 0, i;
178     point pp[MAXN]; //尽量定义成全局变量
179     for(i = 0; i < n; i++) {
180         if(decideSide(p[i], side, l1, l2) == 1) pp[m++] = p[i];
181         if(decideSide(p[i], p[(i + 1) % n], l1, l2) < 1 && !(zero((p[i] - l2) * (l1 - l2\
182 )) && zero((p[(i + 1) % n] - l2) * (l1 - l2))))
183             pp[m++] = intersection(p[i], p[(i + 1) % n], l1, l2);
184     }
185     for(n = i = 0; i < m; i++)
186         if(!i || !zero(pp[i].x - pp[i - 1].x) || !zero(pp[i].y - pp[i - 1].y)) p[n++] = pp\
187 [i];
188     if(zero(p[n - 1].x - p[0].x) && zero(p[n - 1].y - p[0].y)) n--;
189     if(n < 3) n = 0;
190 } //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
191 inline double Seg_area(const point &p1, const point &p2, const point &p0, double R) {
192     point tmp = (p0 - p1).rotate(p2 - p1);
193     double d = -tmp.y, h1 = -tmp.x, h2 = h1 + (p2 - p1).length();
194     if(d >= R || d <= -R) return R * R * (atan2(d, h1) - atan2(d, h2));
195     double dh = sqrt(R * R - d * d);
196     if(h2 < -dh || dh < h1) return R * R * (atan2(d, h1) - atan2(d, h2));
197     double ret = 0;
198     if(h1 < -dh) ret += atan2(d, h1) - atan2(d, -dh);
199     if(h2 > dh) ret += atan2(d, dh) - atan2(d, h2);
200     return ret * R * R + d * (min(h2, dh) - max(h1, -dh));
201 } //圆与线段交的有向面积
202 int graham(int n, point *p, point *ch, bool comEdge = false) {
203     if(n < 3) {
204         for(int i = 0; i < n; i++) ch[i] = p[i];
205         return n;
206     }
207     const double e1 = comEdge ? eps : -eps;
208     int i, j, k;
209     sort(p, p + n);
210     ch[0] = p[0];
211     ch[1] = p[1];
212     for(i = j = 2; i < n; ch[j++] = p[i++]) while(j > 1 && (ch[j - 2] - ch[j - 1]) * (p\
213 i] - ch[j - 1]) > e1) j--;
214     ch[k = j++] = p[n - 2];
215     for(i = n - 3; i > 0; ch[j++] = p[i--]) while(j > k && (ch[j - 2] - ch[j - 1]) * (p\

```

```

216 i] - ch[j - 1]) > e1) j--;
217 while (j > k && (ch[j - 2] - ch[j - 1]) * (ch[0] - ch[j - 1]) > e1) j--;
218 return j;
219 } //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

```

1 #include <cmath>
2 #include <algorithm>
3 using namespace std;
4 const int MAXN = 1000;
5 const double eps = 1e-8;
6 const double PI = atan2(0.0, -1.0);
7 inline double sqr(double x) {
8     return x * x;
9 }
10 inline bool zero(double x) {
11     return (x > 0 ? x : -x) < eps;
12 }
13 inline int sgn(double x) {
14     return (x > eps ? 1 : (x + eps < 0 ? -1 : 0));
15 }
16 struct point3 {
17     double x, y, z;
18     point3(double x, double y, double z):x(x), y(y), z(z) {}
19     point3() {}
20     bool operator == (const point3 &a) const {
21         return sgn(x - a.x) == 0 && sgn(y - a.y) == 0 && sgn(z - a.z) == 0;
22     }
23     bool operator != (const point3 &a) const {
24         return sgn(x - a.x) != 0 || sgn(y - a.y) != 0 || sgn(z - a.z) != 0;
25     }
26     bool operator < (const point3 &a) const {
27         return sgn(x - a.x) < 0 || sgn(x - a.x) == 0 && sgn(y - a.y) < 0 || sgn(x - a.x) \
28 == 0 && sgn(y - a.y) == 0 && sgn(z - a.z) < 0;
29     }
30     point3 operator + (const point3 &a) const {
31         return point3(x + a.x, y + a.y, z + a.z);
32     }
33     point3 operator - (const point3 &a) const {
34         return point3(x - a.x, y - a.y, z - a.z);
35     }
36     point3 operator * (const double &a) const {
37         return point3(x * a, y * a, z * a);
38     }
39     point3 operator / (const double &a) const {
40         return point3(x / a, y / a, z / a);
41     }
42     point3 operator * (const point3 &a) const {
43         return point3(y * a.z - z * a.y, z * a.x - x * a.z, x * a.y - y * a.x); //xmu\
44 Lt
45 }
46 double operator ^ (const point3 &a) const {
47     return x * a.x + y * a.y + z * a.z; //dmult

```



```

48     }
49     double sqrlen() const {
50         return sqr(x) + sqr(y) + sqr(z);
51     }
52     double length() const {
53         return sqrt(sqrlen());
54     }
55     point3 trunc(double a) const {
56         return (*this) * (a / length());
57     }
58     point3 rotate(const point3 &a, const point3 &b, const point3 &c) const {
59         return point3(a ^ (*this), b ^ (*this), c ^ (*this)); //abc 正交且模为 1
60     }
61 };
62 inline point3 pvec(const point3 &a, const point3 &b, const point3 &c) {
63     return (a - b) * (b - c);
64 } //平面法向量
65 inline bool dotsInline(const point3 &a, const point3 &b, const point3 &c) {
66     return zero(((a - b) * (b - c)).length());
67 } //判三点共线
68 inline bool dotsOnplane(const point3 &a, const point3 &b, const point3 &c, const point3 &d) {
69     return zero(pvec(a, b, c) ^ (d - a));
70 } //判四点共面
71 inline bool dotOnlineIn(const point3 &p, const point3 &l1, const point3 &l2) {
72     return zero(((p - l1) * (p - l2)).length()) && (l1.x - p.x) * (l2.x - p.x) < eps && \
73     (l1.y - p.y) * (l2.y - p.y) < eps && (l1.z - p.z) * (l2.z - p.z) < eps;
74 } //判点是否在线段上, 包括端点和共线
75 inline bool dotInplaneIn(const point3 &p, const point3 &a, const point3 &b, const point3 &c) {
76     return zero(((a - b) * (a - c)).length() - ((p - a) * (p - b)).length() - ((p - b) * \
77     (p - c)).length() - ((p - c) * (p - a)).length());
78 } //判点是否在空间三角形上, 包括边界, 须保证 abc 不共线
79 inline int decideSide(const point3 &p1, const point3 &p2, const point3 &l1, const point3 &l2) {
80     return sgn(((l1 - l2) * (p1 - l2)) ^ ((l1 - l2) * (p2 - l2)));
81 } //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
82 inline int decideSide(const point3 &p1, const point3 &p2, const point3 &a, const point3 &b, const point3 &c) {
83     return sgn((pvec(a, b, c) ^ (p1 - a)) * (pvec(a, b, c) ^ (p2 - a)));
84 } //点 p1 和 p2, 平面 abc, -1 表示在异侧, 0 表示在面上, 1 表示同侧
85 inline bool parallel(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &v2) {
86     return zero(((u1 - u2) * (v1 - v2)).length());
87 } //判两直线平行
88 inline bool parallel(const point3 &a, const point3 &b, const point3 &c, const point3 &d, \
89     const point3 &e, const point3 &f) {
90     return zero((pvec(a, b, c) * pvec(d, e, f)).length());
91 } //判两平面平行
92 inline bool parallel(const point3 &l1, const point3 &l2, const point3 &a, const point3 &b, \
93     const point3 &c) {

```

```

99     return zero((l1 - l2) ^ pvec(a, b, c));
100 } //判直线与平面平行
101 inline bool perpendicular(const point3 &u1, const point3 &u2, const point3 &v1, const po\
102 int3 &v2) {
103     return zero((u1 - u2) ^ (v1 - v2));
104 } //判两直线垂直
105 inline bool perpendicular(const point3 &a, const point3 &b, const point3 &c, const point\
106 3 &d, const point3 &e, const point3 &f) {
107     return zero(pvec(a, b, c) ^ pvec(d, e, f));
108 } //判两平面垂直
109 inline bool perpendicular(const point3 &l1, const point3 &l2, const point3 &a, const poi\
110 nt3 &b, const point3 &c) {
111     return zero(((l1 - l2) * pvec(a, b, c)).length());
112 } //判直线与平面垂直
113 inline bool intersectIn(const point3 &u1, const point3 &u2, const point3 &v1, const poin\
114 t3 &v2) {
115     if(!dotsOnplane(u1, u2, v1, v2)) return false;
116     if(!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2))
117         return decideSide(u1, u2, v1, v2) < 1 && decideSide(v1, v2, u1, u2) < 1;
118     return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1, u1, u2)\
119 || dotOnlineIn(v2, u1, u2);
120 } //判两线段相交, 包括端点和部分重合
121 inline bool intersectEx(const point3 &u1, const point3 &u2, const point3 &v1, const poin\
122 t3 &v2) {
123     return dotsOnplane(u1, u2, v1, v2) && decideSide(u1, u2, v1, v2) < 0 && decideSide(v\
124 1, v2, u1, u2) < 0;
125 } //判两线段相交, 不包括端点和部分重合
126 inline bool intersect(const point3 &l1, const point3 &l2, const point3 &a, const point3 \
127 &b, const point3 &c, bool edge = true) {
128     return decideSide(l1, l2, a, b, c) < edge && decideSide(a, b, l1, l2, c) < edge && d\
129 ecideSide(b, c, l1, l2, a) < edge && decideSide(c, a, l1, l2, b) < edge;
130 } //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
131 point3 intersection(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &\
132 v2) {
133     point3 p0 = (u1 - v1) * (v1 - v2), p1 = (u1 - u2) * (v1 - v2);
134     return u1 + (u2 - u1) * (sgn(p0 ^ p1) * sqrt(p0.sqrLen() / p1.sqrLen()));
135 } //计算两直线交点, 须预判直线是否共面和平行
136 point3 intersection(const point3 &l1, const point3 &l2, const point3 &a, const point3 &b\
137 , const point3 &c) {
138     point3 temp = pvec(a, b, c);
139     return l1 + (l2 - l1) * ((temp ^ (a - l1)) / (temp ^ (l2 - l1)));
140 } //计算直线与平面交点, 须预判是否平行, 并保证三点不共线
141 void intersection(const point3 &a, const point3 &b, const point3 &c, const point3 &d, co\
142 nst point3 &e, const point3 &f, point3 &p1, point3 &p2) {
143     p1 = parallel(d, e, a, b, c) ? intersection(e, f, a, b, c) : intersection(d, e, a, b\
144 , c);
145     p2 = parallel(f, d, a, b, c) ? intersection(e, f, a, b, c) : intersection(f, d, a, b\
146 , c);
147 } //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
148 inline double disptoline(const point3 &p, const point3 &l1, const point3 &l2) {
149     return sqrt(((p - l1) * (l2 - l1)).sqrLen() / (l1 - l2).sqrLen());

```

```

150 } //点到直线距离
151 inline point3 ptoline(const point3 &p, const point3 &l1, const point3 &l2) {
152     point3 temp = l2 - l1;
153     return l1 + temp * ((p - l1) ^ temp) / (temp ^ temp);
154 } //点到直线最近点
155 inline double disptoplane(const point3 &p, const point3 &a, const point3 &b, const point3 &c) {
156     point3 temp = pvec(a, b, c);
157     return fabs(temp ^ (p - a)) / temp.length();
158 } //点到平面距离
159 inline point3 ptoplane(const point3 &p, const point3 &a, const point3 &b, const point3 &c) {
160     return intersection(p, p + pvec(a, b, c), a, b, c);
161 } //点到平面最近点
162 inline double dislinetoline(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &v2) {
163     point3 temp = (u1 - u2) * (v1 - v2);
164     return fabs((u1 - v1) ^ temp) / temp.length();
165 } //直线到直线距离
166 void linetoline(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &v2, \
167 point3 &p1, point3 &p2) {
168     point3 ab = u2 - u1, cd = v2 - v1, ac = v1 - u1;
169     p2 = v1 + cd * (((ab ^ cd) * (ac ^ ab) - (ab ^ ab) * (ac ^ cd)) / ((ab ^ ab) * (cd ^ cd) - sqr(ab ^ cd)));
170     p1 = ptoline(p2, u1, u2);
171 } //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
172 inline double angleCos(const point3 &u1, const point3 &u2, const point3 &v1, const point3 &v2) {
173     return ((u1 - u2) ^ (v1 - v2)) / sqrt((u1 - u2).sqrln() * (v1 - v2).sqrln());
174 } //两直线夹角 cos 值
175 inline double angleCos(const point3 &a, const point3 &b, const point3 &c, const point3 &d, const point3 &e, const point3 &f) {
176     point3 p1 = pvec(a, b, c), p2 = pvec(d, e, f);
177     return (p1 ^ p2) / sqrt(p1.sqrln() * p2.sqrln());
178 } //两平面夹角 cos 值
179 inline double angleSin(const point3 &l1, const point3 &l2, const point3 &a, const point3 &b, const point3 &c) {
180     point3 temp = pvec(a, b, c);
181     return ((l1 - l2) ^ temp) / sqrt((l1 - l2).sqrln() * temp.sqrln());
182 } //直线平面夹角 sin 值
183 double angle(double lng1, double lat1, double lng2, double lat2) {
184     double dlng = fabs(lng1 - lng2) * PI / 180;
185     while(dlng >= PI + PI) dlng -= PI + PI;
186     if(dlng > PI) dlng = PI + PI - dlng;
187     lat1 *= PI / 180;
188     lat2 *= PI / 180;
189     return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
190 } //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

```

1 //三维几何函数库
2 #include <cmath>
3
4 const double EPS = 1e-8;
5
6 struct Point3D {
7     double x, y, z;
8 };
9
10 struct Line3D {
11     Point3D a, b;
12 };
13
14 struct Plane {
15     Point3D a, b, c;
16 };
17
18 struct PlaneF {
19     //  $ax + by + cz + d = 0$ 
20     double a, b, c, d;
21 };
22
23 inline bool zero(double x) {
24     return (x > 0 ? x : -x) < EPS;
25 }
26
27 //平方
28 inline double sqr(double d) {
29     return d * d;
30 }
31
32 //计算 cross product  $U \times V$ 
33 inline Point3D xmult(const Point3D &u, const Point3D &v) {
34     Point3D ret;
35     ret.x = u.y * v.z - v.y * u.z;
36     ret.y = u.z * v.x - u.x * v.z;
37     ret.z = u.x * v.y - u.y * v.x;
38     return ret;
39 }
40
41 //计算 dot product  $U \cdot V$ 
42 inline double dmult(const Point3D &u, const Point3D &v) {
43     return u.x * v.x + u.y * v.y + u.z * v.z;
44 }
45
46 //矢量差  $U - V$ 
47 inline Point3D subt(const Point3D &u, const Point3D &v) {
48     Point3D ret;
49     ret.x = u.x - v.x;
50     ret.y = u.y - v.y;
51     ret.z = u.z - v.z;
52     return ret;

```

```

53 }
54
55 //取平面法向量
56 inline Point3D pvec(const Plane &s) {
57     return xmult(subt(s.a, s.b), subt(s.b, s.c));
58 }
59 inline Point3D pvec(const Point3D &s1, const Point3D &s2, const Point3D &s3) {
60     return xmult(subt(s1, s2), subt(s2, s3));
61 }
62 inline Point3D pvec(const PlaneF &p) {
63     Point3D ret;
64     ret.x = p.a;
65     ret.y = p.b;
66     ret.z = p.c;
67     return ret;
68 }
69
70 //两点距离
71 inline double dis(const Point3D &p1, const Point3D &p2) {
72     return sqrt((p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) + (p1.z - p2.z)\
73 )*(p1.z - p2.z));
74 }
75
76 //向量大小
77 inline double vlen(const Point3D &p) {
78     return sqrt(p.x*p.x + p.y*p.y + p.z*p.z);
79 }
80
81 //向量大小的平方
82 inline double sqrlen(const Point3D &p) {
83     return (p.x*p.x + p.y*p.y + p.z*p.z);
84 }
85
86 //判三点共线
87 bool dotsInline(const Point3D &p1, const Point3D &p2, const Point3D &p3) {
88     return sqrlen(xmult(subt(p1, p2), subt(p2, p3))) < EPS;
89 }
90
91 //判四点共面
92 bool dotsOnplane(const Point3D &a, const Point3D &b, const Point3D &c, const Point3D &d)\
93 {
94     return zero(dmult(pvec(a, b, c), subt(d, a)));
95 }
96
97 //判点是否在线段上, 包括端点和共线
98 bool dotOnlineIn(const Point3D &p, const Line3D &l) {
99     return zero(sqrlen(xmult(subt(p, l.a), subt(p, l.b)))) && (l.a.x - p.x) * (l.b.x - p\
100 .x) < EPS && (l.a.y - p.y) * (l.b.y - p.y) < EPS && (l.a.z - p.z) * (l.b.z - p.z) < EPS;
101 }
102 bool dotOnlineIn(const Point3D &p, const Point3D &l1, const Point3D &l2) {
103     return zero(sqrlen(xmult(subt(p, l1), subt(p, l2)))) && (l1.x - p.x) * (l2.x - p.x) \
104 < EPS && (l1.y - p.y) * (l2.y - p.y) < EPS && (l1.z - p.z) * (l2.z - p.z) < EPS;
105 }

```

```

106
107 //判点是否在线段上, 不包括端点
108 bool dotOnlineEx(const Point3D &p, const Line3D &l) {
109     return dotOnlineIn(p, l) && (!zero(p.x - l.a.x) || !zero(p.y - l.a.y) || !zero(p.z - \
110 l.a.z)) && (!zero(p.x - l.b.x) || !zero(p.y - l.b.y) || !zero(p.z - l.b.z));
111 }
112 bool dotOnlineEx(const Point3D &p, const Point3D &l1, const Point3D &l2) {
113     return dotOnlineIn(p, l1, l2) && (!zero(p.x - l1.x) || !zero(p.y - l1.y) || !zero(p.\
114 z - l1.z)) && (!zero(p.x - l2.x) || !zero(p.y - l2.y) || !zero(p.z - l2.z));
115 }
116
117 //判点是否在空间三角形上, 包括边界, 三点共线无意义
118 bool dotInplaneIn(const Point3D &p, const Plane &s) {
119     return zero(vlen(xmult(subt(s.a, s.b), subt(s.a, s.c))) - vlen(xmult(subt(p, s.a), s\
120 ubt(p, s.b))) - vlen(xmult(subt(p, s.b), subt(p, s.c))) - vlen(xmult(subt(p, s.c), subt(\
121 p, s.a))));
122 }
123 bool dotInplaneIn(const Point3D &p, const Point3D &s1, const Point3D &s2, const Point3D \
124 &s3) {
125     return zero(vlen(xmult(subt(s1, s2), subt(s1, s3))) - vlen(xmult(subt(p, s1), subt(p\
126 , s2))) - vlen(xmult(subt(p, s2), subt(p, s3))) - vlen(xmult(subt(p, s3), subt(p, s1))))\
127 ;
128 }
129
130 //判点是否在空间三角形上, 不包括边界, 三点共线无意义
131 bool dotInplaneEx(const Point3D &p, const Plane &s) {
132     return dotInplaneIn(p, s) && sqrtlen(xmult(subt(p, s.a), subt(p, s.b))) > EPS && sqrt\
133 en(xmult(subt(p, s.b), subt(p, s.c))) > EPS && sqrtlen(xmult(subt(p, s.c), subt(p, s.a)))\
134 > EPS;
135 }
136 bool dotInplaneEx(const Point3D &p, const Point3D &s1, const Point3D &s2, const Point3D \
137 &s3) {
138     return dotInplaneIn(p, s1, s2, s3) && sqrtlen(xmult(subt(p, s1), subt(p, s2))) > EPS \
139 && sqrtlen(xmult(subt(p, s2), subt(p, s3))) > EPS && sqrtlen(xmult(subt(p, s3), subt(p, s1\
140 ))) > EPS;
141 }
142
143 //判两点在线段同侧, 点在线段上返回 0, 不共面无意义
144 bool sameSide(const Point3D &p1, const Point3D &p2, const Line3D &l) {
145     return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p2, l.\
146 b))) > EPS;
147 }
148 bool sameSide(const Point3D &p1, const Point3D &p2, const Point3D &l1, const Point3D &l2\
149 ) {
150     return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2))) >\
151 EPS;
152 }
153
154 //判两点在线段异侧, 点在线段上返回 0, 不共面无意义
155 bool oppositeSide(const Point3D &p1, const Point3D &p2, const Line3D &l) {
156     return dmult(xmult(subt(l.a, l.b), subt(p1, l.b)), xmult(subt(l.a, l.b), subt(p2, l.\
157 b))) < -EPS;
158 }
159 bool oppositeSide(const Point3D &p1, const Point3D &p2, const Point3D &l1, const Point3D \

```

```

160  &l2) {
161      return dmult(xmult(subt(l1, l2), subt(p1, l2)), xmult(subt(l1, l2), subt(p2, l2))) < \
162      -EPS;
163  }
164
165  //判两点在平面同侧, 点在平面上返回 0
166  bool sameSide(const Point3D &p1, const Point3D &p2, const Plane &s) {
167      return dmult(pvec(s), subt(p1, s.a)) * dmult(pvec(s), subt(p2, s.a)) > EPS;
168  }
169  bool sameSide(const Point3D &p1, const Point3D &p2, const Point3D &s1, const Point3D &s2 \
170  , const Point3D &s3) {
171      return dmult(pvec(s1, s2, s3), subt(p1, s1)) * dmult(pvec(s1, s2, s3), subt(p2, s1)) \
172      > EPS;
173  }
174  bool sameSide(const Point3D &p1, const Point3D &p2, const PlaneF &s) {
175      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 \
176      * p2.z + s.d) > EPS;
177  }
178
179  //判两点在平面异侧, 点在平面上返回 0
180  bool oppositeSide(const Point3D &p1, const Point3D &p2, const Plane &s) {
181      return dmult(pvec(s), subt(p1, s.a)) * dmult(pvec(s), subt(p2, s.a)) < -EPS;
182  }
183  bool oppositeSide(const Point3D &p1, const Point3D &p2, const Point3D &s1, const Point3D \
184  &s2, const Point3D &s3) {
185      return dmult(pvec(s1, s2, s3), subt(p1, s1)) * dmult(pvec(s1, s2, s3), subt(p2, s1)) \
186      < -EPS;
187  }
188  bool oppositeSide(const Point3D &p1, const Point3D &p2, const PlaneF &s) {
189      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;
190  }
191
192  //判两直线平行
193  bool parallel(const Line3D &u, const Line3D &v) {
194      return sqrlen(xmult(subt(u.a, u.b), subt(v.a, v.b))) < EPS;
195  }
196  bool parallel(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D &v2 \
197  ) {
198      return sqrlen(xmult(subt(u1, u2), subt(v1, v2))) < EPS;
199  }
200
201  //判两平面平行
202  bool parallel(const Plane &u, const Plane &v) {
203      return sqrlen(xmult(pvec(u), pvec(v))) < EPS;
204  }
205  bool parallel(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point3D &v1 \
206  , const Point3D &v2, const Point3D &v3) {
207      return sqrlen(xmult(pvec(u1, u2, u3), pvec(v1, v2, v3))) < EPS;
208  }
209  bool parallel(const PlaneF &u, const PlaneF &v) {
210      return sqrlen(xmult(pvec(u), pvec(v))) < EPS;
211  }
212
213  //判直线与平面平行
214  bool parallel(const Line3D &l, const Plane &s) {

```

```

215     return zero(dmult(subt(l.a, l.b), pvec(s)));
216 }
217 bool parallel(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D &s2\
218 , const Point3D &s3) {
219     return zero(dmult(subt(l1, l2), pvec(s1, s2, s3)));
220 }
221 bool parallel(const Line3D &l, const PlaneF &s) {
222     return zero(dmult(subt(l.a, l.b), pvec(s)));
223 }
224
225 //判两直线垂直
226 bool perpendicular(const Line3D &u, const Line3D &v) {
227     return zero(dmult(subt(u.a, u.b), subt(v.a, v.b)));
228 }
229 bool perpendicular(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D\
230 &v2) {
231     return zero(dmult(subt(u1, u2), subt(v1, v2)));
232 }
233
234 //判两平面垂直
235 bool perpendicular(const Plane &u, const Plane &v) {
236     return zero(dmult(pvec(u), pvec(v)));
237 }
238 bool perpendicular(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point3D\
239 &v1, const Point3D &v2, const Point3D &v3) {
240     return zero(dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)));
241 }
242 bool perpendicular(const PlaneF &u, const PlaneF &v) {
243     return zero(dmult(pvec(u), pvec(v)));
244 }
245
246 //判直线与平面垂直
247 bool perpendicular(const Line3D &l, const Plane &s) {
248     return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < EPS;
249 }
250 bool perpendicular(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D\
251 &s2, const Point3D &s3) {
252     return sqrlen(xmult(subt(l1, l2), pvec(s1, s2, s3))) < EPS;
253 }
254 bool perpendicular(const Line3D &l, const PlaneF &s) {
255     return sqrlen(xmult(subt(l.a, l.b), pvec(s))) < EPS;
256 }
257
258 //判两线段相交，包括端点和部分重合
259 bool intersectIn(const Line3D &u, const Line3D &v) {
260     if (!dotsOnplane(u.a, u.b, v.a, v.b)) {
261         return 0;
262     } else if (!dotsInline(u.a, u.b, v.a) || !dotsInline(u.a, u.b, v.b)) {
263         return !sameSide(u.a, u.b, v) && !sameSide(v.a, v.b, u);
264     } else {
265         return dotOnlineIn(u.a, v) || dotOnlineIn(u.b, v) || dotOnlineIn(v.a, u) || dotO\
266 nlineIn(v.b, u);
267     }
268 }
269 bool intersectIn(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D \

```



```

270 &v2) {
271     if (!dotsOnplane(u1, u2, v1, v2)) {
272         return 0;
273     } else if (!dotsInline(u1, u2, v1) || !dotsInline(u1, u2, v2)) {
274         return !sameSide(u1, u2, v1, v2) && !sameSide(v1, v2, u1, u2);
275     } else {
276         return dotOnlineIn(u1, v1, v2) || dotOnlineIn(u2, v1, v2) || dotOnlineIn(v1, u1, \
277 u2) || dotOnlineIn(v2, u1, u2);
278     }
279 }
280
281 //判两线段相交, 不包括端点和部分重合
282 bool intersectEx(const Line3D &u, const Line3D &v) {
283     return dotsOnplane(u.a, u.b, v.a, v.b) && oppositeSide(u.a, u.b, v) && oppositeSide(\
284 v.a, v.b, u);
285 }
286 bool intersectEx(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D \
287 &v2) {
288     return dotsOnplane(u1, u2, v1, v2) && oppositeSide(u1, u2, v1, v2) && oppositeSide(v\
289 1, v2, u1, u2);
290 }
291
292 //判线段与空间三角形相交, 包括交于边界和 (部分) 包含
293 bool intersectIn(const Line3D &l, const Plane &s) {
294     return !sameSide(l.a, l.b, s) && !sameSide(s.a, s.b, l.a, l.b, s.c) && !sameSide(s.b\
295 , s.c, l.a, l.b, s.a) && !sameSide(s.c, s.a, l.a, l.b, s.b);
296 }
297 bool intersectIn(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D \
298 &s2, const Point3D &s3) {
299     return !sameSide(l1, l2, s1, s2, s3) && !sameSide(s1, s2, l1, l2, s3) && !sameSide(s\
300 2, s3, l1, l2, s1) && !sameSide(s3, s1, l1, l2, s2);
301 }
302
303 //判线段与空间三角形相交, 不包括交于边界和 (部分) 包含
304 bool intersectEx(const Line3D &l, const Plane &s) {
305     return oppositeSide(l.a, l.b, s) && oppositeSide(s.a, s.b, l.a, l.b, s.c) && opposit\
306 eSide(s.b, s.c, l.a, l.b, s.a) && oppositeSide(s.c, s.a, l.a, l.b, s.b);
307 }
308 bool intersectEx(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D \
309 &s2, const Point3D &s3) {
310     return oppositeSide(l1, l2, s1, s2, s3) && oppositeSide(s1, s2, l1, l2, s3) && oppos\
311 iteSide(s2, s3, l1, l2, s1) && oppositeSide(s3, s1, l1, l2, s2);
312 }
313
314 //计算两直线交点, 注意事先判断直线是否共面和平行!
315 //线段交点请另外判线段相交 (同时还是要判断是否平行!)
316 #include <algorithm>
317
318 using namespace std;
319 Point3D intersection(Point3D u1, Point3D u2, Point3D v1, Point3D v2) {
320     double dxu = u2.x - u1.x;
321     double dyu = u2.y - u1.y;
322     double dzu = u2.z - u1.z;
323     double dxv = v2.x - v1.x;
324     double dyv = v2.y - v1.y;

```

```

325     double dzv = v2.z - v1.z;
326     double t;
327     if (!zero(dxu * dyv - dyu * dxv)) {
328         t = (dyv * (v1.x - u1.x) + dxv * (u1.y - v1.y)) / (dxu * dyv - dyu * dxv);
329     } else if (!zero(dxu * dzv - dzu * dxv)) {
330         t = (dzv * (v1.x - u1.x) + dxv * (u1.z - v1.z)) / (dxu * dzv - dzu * dxv);
331     } else {
332         t = (dzv * (v1.y - u1.y) + dyv * (u1.z - v1.z)) / (dyu * dzv - dzu * dyv);
333     }
334     Point3D ret;
335     ret.x = u1.x + dxu * t;
336     ret.y = u1.y + dyu * t;
337     ret.z = u1.z + dzu * t;
338     return ret;
339 }
340
341 //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!
342 //线段和空间三角形交点请另外判断
343 Point3D intersection(const Line3D &l, const Plane &s) {
344     Point3D ret = pvec(s);
345     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\
346 .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));
347     ret.x = l.a.x + (l.b.x - l.a.x) * t;
348     ret.y = l.a.y + (l.b.y - l.a.y) * t;
349     ret.z = l.a.z + (l.b.z - l.a.z) * t;
350     return ret;
351 }
352 Point3D intersection(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Poin\
353 t3D &s2, const Point3D &s3) {
354     Point3D ret = pvec(s1, s2, s3);
355     double t = (ret.x * (s1.x - l1.x) + ret.y * (s1.y - l1.y) + ret.z * (s1.z - l1.z)) /\
356 (ret.x * (l2.x - l1.x) + ret.y * (l2.y - l1.y) + ret.z * (l2.z - l1.z));
357     ret.x = l1.x + (l2.x - l1.x) * t;
358     ret.y = l1.y + (l2.y - l1.y) * t;
359     ret.z = l1.z + (l2.z - l1.z) * t;
360     return ret;
361 }
362 Point3D intersection(const Line3D &l, const PlaneF &s) {
363     Point3D ret = subt(l.b, l.a);
364     double t = -(dmult(pvec(s), l.a) + s.d) / (dmult(pvec(s), ret));
365     ret.x = ret.x * t + l.a.x;
366     ret.y = ret.y * t + l.a.y;
367     ret.z = ret.z * t + l.a.z;
368     return ret;
369 }
370
371 //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!
372 Line3D intersection(const Plane &u, const Plane &v) {
373     Line3D ret;
374     ret.a = parallel(v.a, v.b, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.c) : \
375 intersection(v.a, v.b, u.a, u.b, u.c);
376     ret.b = parallel(v.c, v.a, u.a, u.b, u.c) ? intersection(v.b, v.c, u.a, u.b, u.c) : \
377 intersection(v.c, v.a, u.a, u.b, u.c);
378     return ret;
379 }
380 Line3D intersection(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point\

```

```

381 3D &v1, const Point3D &v2, const Point3D &v3) {
382     Line3D ret;
383     ret.a = parallel(v1, v2, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : intersecti\
384 on(v1, v2, u1, u2, u3);
385     ret.b = parallel(v3, v1, u1, u2, u3) ? intersection(v2, v3, u1, u2, u3) : intersecti\
386 on(v3, v1, u1, u2, u3);
387     return ret;
388 }
389
390 //点到直线距离
391 double disptoline(const Point3D &p, const Line3D &l) {
392     return vlen(xmult(subt(p, l.a), subt(l.b, l.a))) / dis(l.a, l.b);
393 }
394 double disptoline(const Point3D &p, const Point3D &l1, const Point3D &l2) {
395     return vlen(xmult(subt(p, l1), subt(l2, l1))) / dis(l1, l2);
396 }
397
398 //点到直线最近点
399 Point3D ptoline(const Point3D &p, const Line3D &l) {
400     Point3D ab = subt(l.b, l.a);
401     double t = - dmult(subt(p, l.a), ab) / sqrlen(ab);
402     ab.x *= t;
403     ab.y *= t;
404     ab.z *= t;
405     return subt(l.a, ab);
406 }
407
408 //点到平面距离
409 double disptoplane(const Point3D &p, const Plane &s) {
410     return fabs(dmult(pvec(s), subt(p, s.a))) / vlen(pvec(s));
411 }
412 double disptoplane(const Point3D &p, const Point3D &s1, const Point3D &s2, const Point3D\
413 &s3) {
414     return fabs(dmult(pvec(s1, s2, s3), subt(p, s1))) / vlen(pvec(s1, s2, s3));
415 }
416 double disptoplane(const Point3D &p, const PlaneF &s) {
417     return fabs((dmult(pvec(s), p)+s.d) / vlen(pvec(s)));
418 }
419
420 //点到平面最近点
421 Point3D ptoplane(const Point3D &p, const PlaneF &s) {
422     Line3D l;
423     l.a = p;
424     l.b = pvec(s);
425     l.b.x += p.x;
426     l.b.y += p.y;
427     l.b.z += p.z;
428     return intersection(l, s);
429 }
430
431 //直线到直线距离
432 double dislinetoline(const Line3D &u, const Line3D &v) {
433     Point3D n = xmult(subt(u.a, u.b), subt(v.a, v.b));
434     return fabs(dmult(subt(u.a, v.a), n)) / vlen(n);
435 }

```

```

436 double dislinetoline(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Poin\
437 t3D &v2) {
438     Point3D n = xmult(subt(u1, u2), subt(v1, v2));
439     return fabs(dmult(subt(u1, v1), n)) / vlen(n);
440 }
441
442 //直线到直线的最近点对
443 //p1 在 u 上, p2 在 v 上, p1 到 p2 是 uv 之间的最近距离
444 //注意, 保证两直线不平行
445 void linetoline(const Line3D &u, const Line3D &v, Point3D &p1, Point3D &p2) {
446     Point3D ab = subt(u.b, u.a), cd = subt(v.b, v.a), ac = subt(v.a, u.a);
447     double r = (dmult(ab, cd) * dmult(ac, ab) - sqrlen(ab) * dmult(ac, cd)) / (sqrlen(ab)\
448 ) * sqrlen(cd) - sqr(dmult(ab, cd)));
449     p2.x = v.a.x + r * cd.x;
450     p2.y = v.a.y + r * cd.y;
451     p2.z = v.a.z + r * cd.z;
452     p1 = ptoline(p2, u);
453 }
454
455 //两直线夹角 cos 值
456 double angleCos(const Line3D &u, const Line3D &v) {
457     return dmult(subt(u.a, u.b), subt(v.a, v.b)) / vlen(subt(u.a, u.b)) / vlen(subt(v.a,\
458 v.b));
459 }
460 double angleCos(const Point3D &u1, const Point3D &u2, const Point3D &v1, const Point3D &\
461 v2) {
462     return dmult(subt(u1, u2), subt(v1, v2)) / vlen(subt(u1, u2)) / vlen(subt(v1, v2));
463 }
464
465 //两平面夹角 cos 值
466 double angleCos(const Plane &u, const Plane &v) {
467     return dmult(pvec(u), pvec(v)) / vlen(pvec(u)) / vlen(pvec(v));
468 }
469 double angleCos(const Point3D &u1, const Point3D &u2, const Point3D &u3, const Point3D &\
470 v1, const Point3D &v2, const Point3D &v3) {
471     return dmult(pvec(u1, u2, u3), pvec(v1, v2, v3)) / vlen(pvec(u1, u2, u3)) / vlen(pve\
472 c(v1, v2, v3));
473 }
474 double angleCos(const PlaneF &u, const PlaneF &v) {
475     return dmult(pvec(u), pvec(v)) / (vlen(pvec(u)) * vlen(pvec(v)));
476 }
477
478 //直线平面夹角 sin 值
479 double angleSin(const Line3D &l, const Plane &s) {
480     return dmult(subt(l.a, l.b), pvec(s)) / vlen(subt(l.a, l.b)) / vlen(pvec(s));
481 }
482 double angleSin(const Point3D &l1, const Point3D &l2, const Point3D &s1, const Point3D &\
483 s2, const Point3D &s3) {
484     return dmult(subt(l1, l2), pvec(s1, s2, s3)) / vlen(subt(l1, l2)) / vlen(pvec(s1, s2\
485 , s3));
486 }
487 double angleSin(Line3D l, const PlaneF &s) {
488     return dmult(subt(l.a, l.b), pvec(s)) / (vlen(subt(l.a, l.b)) * vlen(pvec(s)));
489 }

```

```

490 |
491 | // 平面方程形式转化 Plane -> PlaneF
492 | PlaneF planeToPlaneF(const Plane &p) {
493 |     PlaneF ret;
494 |     Point3D m = xmult(subt(p.b, p.a), subt(p.c, p.a));
495 |     ret.a = m.x;
496 |     ret.b = m.y;
497 |     ret.c = m.z;
498 |     ret.d = -m.x * p.a.x - m.y * p.a.y - m.z * p.a.z;
499 |     return ret;
500 | }

```

1.7 三角形

```

1 | #include <cmath>
2 |
3 | struct Point {
4 |     double x, y;
5 | };
6 | struct Line {
7 |     Point a, b;
8 | };
9 |
10 | inline double dis(const Point &p1, const Point &p2) {
11 |     return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
12 | }
13 |
14 | Point intersection(const Line &u, const Line &v) {
15 |     Point ret = u.a;
16 |     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)) /\
17 | ((u.a.x - u.b.x) * (v.a.y - v.b.y) - (u.a.y - u.b.y) * (v.a.x - v.b.x));
18 |     ret.x += (u.b.x - u.a.x) * t;
19 |     ret.y += (u.b.y - u.a.y) * t;
20 |     return ret;
21 | }
22 |
23 | //外心
24 | Point circumcenter(const Point &a, const Point &b, const Point &c) {
25 |     Line u, v;
26 |     u.a.x = (a.x + b.x) / 2;
27 |     u.a.y = (a.y + b.y) / 2;
28 |     u.b.x = u.a.x - a.y + b.y;
29 |     u.b.y = u.a.y + a.x - b.x;
30 |     v.a.x = (a.x + c.x) / 2;
31 |     v.a.y = (a.y + c.y) / 2;
32 |     v.b.x = v.a.x - a.y + c.y;
33 |     v.b.y = v.a.y + a.x - c.x;
34 |     return intersection(u, v);
35 | }
36 |
37 | //内心
38 | Point incenter(const Point &a, const Point &b, const Point &c) {
39 |     Line u, v;
40 |     double m, n;
41 |     u.a = a;

```

```

42     m = atan2(b.y - a.y, b.x - a.x);
43     n = atan2(c.y - a.y, c.x - a.x);
44     u.b.x = u.a.x + cos((m + n) / 2);
45     u.b.y = u.a.y + sin((m + n) / 2);
46     v.a = b;
47     m = atan2(a.y - b.y, a.x - b.x);
48     n = atan2(c.y - b.y, c.x - b.x);
49     v.b.x = v.a.x + cos((m + n) / 2);
50     v.b.y = v.a.y + sin((m + n) / 2);
51     return intersection(u, v);
52 }
53
54 //垂心
55 Point perpercenter(const Point &a, const Point &b, const Point &c) {
56     Line u, v;
57     u.a = c;
58     u.b.x = u.a.x - a.y + b.y;
59     u.b.y = u.a.y + a.x - b.x;
60     v.a = b;
61     v.b.x = v.a.x - a.y + c.y;
62     v.b.y = v.a.y + a.x - c.x;
63     return intersection(u, v);
64 }
65
66 //重心
67 //到三角形三顶点距离的平方和最小的点
68 //三角形内到三边距离之积最大的点
69 Point barycenter(const Point &a, const Point &b, const Point &c) {
70     Line u, v;
71     u.a.x = (a.x + b.x) / 2;
72     u.a.y = (a.y + b.y) / 2;
73     u.b = c;
74     v.a.x = (a.x + c.x) / 2;
75     v.a.y = (a.y + c.y) / 2;
76     v.b = b;
77     return intersection(u, v);
78 }

```

1.8 任意维空间最近点对

```

1 // 任意维最近点对
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef pair<int,int> pii;
5
6 const double EPS=1e-9, INF=1e9;
7 const int DIM=3; // 点集的维数
8
9 template <class T> T sqr(T x) {
10     return x*x;
11 }
12
13 int fcmp(double x) {

```

```

14     return fabs(x)<EPS ? 0 : (x<0 ? -1 : 1);
15 }
16
17 struct Point {
18     double x[DIM];
19     int id;
20     double &operator[](int i) {
21         return x[i];
22     }
23     friend double dis(Point a, Point b) {
24         double res = 0;
25         for(int i = 0; i < DIM; ++i)
26             res += sqr(a[i] - b[i]);
27         return sqrt(res);
28     }
29 };
30
31 struct Comp {
32     int dim;
33     bool operator()(Point L, Point R) const {
34         return L.x[dim] < R.x[dim];
35     }
36     Comp(int dim): dim(dim) {}
37 };
38
39 // 用 work() 函数找最近点对, 从 min_dis 和 best_pair 中找到你需要的信息
40 // 复杂度约为  $O(n*(\log(n))^{\text{DIM}})$ 
41 struct ClosestPair {
42     double min_dis; // 储存最近距离
43     pii best_pair; // 储存最近点对的标号
44     vector<Point> vtmp[DIM];
45
46     void init() {
47         min_dis = INF;
48         best_pair = pii(-1, -1);
49         for(int i=0; i<DIM; ++i)
50             vtmp[i].clear();
51     }
52
53     // 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
54     // 请按实际需要修改
55     void take_best(Point a, Point b) {
56         if (a.id > b.id) swap(a, b);
57         int cmp_res = fcmp(dis(a, b) - min_dis);
58         if (
59             best_pair.first == -1 || cmp_res < 0 ||
60             (cmp_res == 0 && pii(a.id, b.id) < best_pair)
61         ) {
62             best_pair = pii(a.id, b.id);
63             min_dis = dis(a, b);
64         }
65     }
66
67     // 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

```

```

68 void closest_pair(vector<Point> &v, int L, int R, int dim=0) {
69     if (R-L <= 6) {
70         for (int i=L+1; i<=R; ++i)
71             for (int j=L; j<i; ++j)
72                 take_best(v[i], v[j]);
73         return;
74     }
75
76     if (dim+1 == DIM) {
77         int z = dim - 1;
78         for (int i=L; i<R; ++i) {
79             for (int j=i+1; j<=R && v[j][z]-v[i][z]<min_dis; ++j)
80                 take_best(v[i], v[j]);
81         }
82         return;
83     }
84
85     int M = (L+R) / 2;
86     closest_pair(v, L, M, dim);
87     closest_pair(v, M+1, R, dim);
88
89     vector<Point> &u = vtmp[dim];
90     u.clear();
91     for(int i=L; i<=R; ++i)
92         if (fcmp(abs(v[i][dim]-v[M][dim]) - min_dis) <= 0) {
93             u.push_back(v[i]);
94         }
95
96     sort(u.begin(), u.end(), Comp(dim+1));
97     if(!u.empty())closest_pair(u, 0, (int)u.size()-1, dim+1);
98 }
99 public:
100 double work(vector<Point> &v) {
101     init();
102     sort(v.begin(), v.end(), Comp(0));
103     closest_pair(v, 0, (int)v.size()-1);
104     return min_dis;
105 }
106 } closest_pair;

```

1.9 圆

```

1 #include <cmath>
2
3 const double EPS = 1e-8;
4
5 struct Point {
6     double x, y;
7 };
8
9 inline double xmult(const Point &p1, const Point &p2, const Point &p0) {
10     return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
11 }
12
13 inline double dis(const Point &p1, const Point &p2) {
14     return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
15 }

```



```

16
17 double disptoline(const Point &p, const Point &l1, const Point &l2) {
18     return fabs(xmult(p, l1, l2)) / dis(l1, l2);
19 }
20
21 Point intersection(const Point &u1, const Point &u2, const Point &v1, const Point &v2) {
22     Point ret = u1;
23     double t = ((u1.x - v1.x) * (v1.y - v2.y) - (u1.y - v1.y) * (v1.x - v2.x)) / ((u1.x \
24 - u2.x) * (v1.y - v2.y) - (u1.y - u2.y) * (v1.x - v2.x));
25     ret.x += (u2.x - u1.x) * t;
26     ret.y += (u2.y - u1.y) * t;
27     return ret;
28 }
29
30 //判直线和圆相交, 包括相切
31 int intersectLineCircle(const Point &c, double r, const Point &l1, const Point &l2) {
32     return disptoline(c, l1, l2) < r + EPS;
33 }
34
35 //判线段和圆相交, 包括端点和相切
36 int intersectSegCircle(const Point &c, double r, const Point &l1, const Point &l2) {
37     double t1 = dis(c, l1) - r, t2 = dis(c, l2) - r;
38     Point t = c;
39     if (t1 < EPS || t2 < EPS) {
40         return t1 > -EPS || t2 > -EPS;
41     }
42     t.x += l1.y - l2.y;
43     t.y += l2.x - l1.x;
44     return xmult(l1, c, t) * xmult(l2, c, t) < EPS && disptoline(c, l1, l2) - r < EPS;
45 }
46
47 //判圆和圆相交, 包括相切
48 int intersectCircleCircle(const Point &c1, double r1, const Point &c2, double r2) {
49     return dis(c1, c2) < r1 + r2 + EPS && dis(c1, c2) > fabs(r1 - r2) - EPS;
50 }
51
52 //计算圆上到点 p 最近点, 如 p 与圆心重合, 返回 p 本身
53 Point dotToCircle(const Point &c, double r, const Point &p) {
54     Point u, v;
55     if (dis(p, c) < EPS) {
56         return p;
57     }
58     u.x = c.x + r * fabs(c.x - p.x) / dis(c, p);
59     u.y = c.y + r * fabs(c.y - p.y) / dis(c, p) * ((c.x - p.x) * (c.y - p.y) < 0 ? -1 : \
60 1);
61     v.x = c.x - r * fabs(c.x - p.x) / dis(c, p);
62     v.y = c.y - r * fabs(c.y - p.y) / dis(c, p) * ((c.x - p.x) * (c.y - p.y) < 0 ? -1 : \
63 1);
64     return dis(u, p) < dis(v, p) ? u : v;
65 }
66
67 //计算直线与圆的交点, 保证直线与圆有交点
68 //计算线段与圆的交点可用这个函数后判点是否在线段上
69 void intersectionLineCircle(const Point &c, double r, const Point &l1, const Point &l2, \

```

```

70 Point &p1, Point &p2) {
71     Point p = c;
72     p.x += l1.y - l2.y;
73     p.y += l2.x - l1.x;
74     p = intersection(p, c, l1, l2);
75     double t = sqrt(r * r - dis(p, c) * dis(p, c)) / dis(l1, l2);
76     p1.x = p.x + (l2.x - l1.x) * t;
77     p1.y = p.y + (l2.y - l1.y) * t;
78     p2.x = p.x - (l2.x - l1.x) * t;
79     p2.y = p.y - (l2.y - l1.y) * t;
80 }
81
82 //计算圆与圆的交点, 保证圆与圆有交点, 圆心不重合
83 void intersectionCircleCircle(const Point &c1, double r1, const Point &c2, double r2, Po\
84 int &p1, Point &p2) {
85     Point u, v;
86     double t = (1 + (r1 * r1 - r2 * r2) / dis(c1, c2) / dis(c1, c2)) / 2;
87     u.x = c1.x + (c2.x - c1.x) * t;
88     u.y = c1.y + (c2.y - c1.y) * t;
89     v.x = u.x + c1.y - c2.y;
90     v.y = u.y - c1.x + c2.x;
91     intersectionLineCircle(c1, r1, u, v, p1, p2);
92 }

```

1.10 圆并

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 //计算  $n(n \leq \text{MAXN})$  个圆形面积的并, 并给出其质心坐标 (cx, cy)
4 //坐标和半径绝对值超过 10000 时需要修改避免爆 int
5 //用 add_circle(x, y, r) 添加圆
6 //多 case 时用 clear() 清空
7 //复杂度  $\mathcal{O}(n^2 \log n)$ 
8 template<class T> T sqr(T x) {
9     return x*x;
10 }
11 typedef double flt;
12 const flt EPS=1e-9;
13 const int MAXN=2001;
14 const flt PI=acos(-1.0);
15 struct CIRU {
16     int x[MAXN],y[MAXN],r[MAXN],n;
17     void clear() {
18         n=0;
19     }
20     void add_circle(int xx,int yy,int rr) {
21         x[n]=xx,y[n]=yy,r[n++]=rr;
22     }
23     int sqdis(int i, int j) {
24         return sqr(x[i]-x[j])+sqr(y[i]-y[j]);
25     }
26     flt fix(flt x) {
27         if(x<-PI)x+=2*PI;

```

```

28     if(x>PI)x-=2*PI;
29     return x;
30 }
31 pair<flt,int> v[MAXN*4+9];
32 flt cx, cy;
33 flt calc_area() {
34     flt area=0;
35     cx=cy=0;
36     for(int i=0; i<n; ++i) {
37         bool ok=1;
38         int tot=2;
39         v[0]=make_pair(-PI,0);
40         v[1]=make_pair(PI,0);
41         for(int j=0; j<n; ++j)if(j!=i) {
42             if(x[j]==x[i]&&y[j]==y[i]&&r[j]==r[i]&&j<i) {
43                 ok=0;
44                 break;
45             }
46             int dis=sqdis(i,j);
47             if(r[i]<r[j]&&dis<=sqr(r[i]-r[j])) {
48                 ok=0;
49                 break;
50             }
51             if(dis>=sqr(r[i]+r[j])||dis<=sqr(r[i]-r[j]))continue;
52             flt delta = (sqr(r[i])-sqr(r[j])+dis)*0.5;
53             delta = acos(delta / (sqrt(flt(dis))*r[i]));
54             flt dir = atan2(flt(y[j]-y[i]),flt(x[j]-x[i]));
55             flt bg=dir-delta, ed=dir+delta;
56             bg=fix(bg),ed=fix(ed);
57             v[tot++]=make_pair(bg,1);
58             v[tot++]=make_pair(ed,-1);
59             if(bg>ed+EPS) {
60                 v[tot++]=make_pair(PI,-1);
61                 v[tot++]=make_pair(-PI,1);
62             }
63         }
64         if(!ok)continue;
65         if(tot==2) {
66             flt s=2*PI*sqr(r[i]);
67             area+=s; //与其他圆不相交
68             cx+=s*x[i];
69             cy+=s*y[i];
70         } else {
71             sort(v,v+tot);
72             int cnt=0;
73             for(int j=0; j+1<tot; ++j) {
74                 cnt+=v[j].second;
75                 flt a1=v[j].first,a2=v[j+1].first;
76                 if(cnt==0 && a2-a1>EPS) {
77                     //只算面积时可以只用下一行注释内的替换后面的所有
78                     //area += r[i] * ((a2-a1)*r[i] + y[i]*(cos(a1)-cos(a2)) + x[i]*(\
79 sin(a2)-sin(a1)));
80                 flt x1=x[i]+r[i]*cos(a1), y1=y[i]+r[i]*sin(a1);
81                 flt x2=x[i]+r[i]*cos(a2), y2=y[i]+r[i]*sin(a2);
82                 flt s_poly = x1*y2-x2*y1; //被包围的多边形面积

```

```

83         flt s_fan = (a2-a1)*sqr(r[i]); //扇形面积
84         flt s_tri = -sin(a2-a1)*sqr(r[i]); //扇形内的三角形面积，注意是 \
85         负的
86         area+=s_poly+s_fan+s_tri;
87         flt sx=4.0/3.0*sin((a2-a1)/2)*r[i]*r[i]*r[i]/s_fan; //以下是求质 \
88         心的
89         cx += (x1+x2)*s_poly/3;
90         cy += (y1+y2)*s_poly/3;
91         cx += (x[i]+sx*cos((a1+a2)/2))*s_fan;
92         cy += (y[i]+sx*sin((a1+a2)/2))*s_fan;
93         cx += (x1+x2+x[i])*s_tri/3;
94         cy += (y1+y2+y[i])*s_tri/3;
95     }
96 }
97 }
98 }
99 if(fabs(area)>EPS) {
100     cx/=area;
101     cy/=area;
102 }
103 return area*0.5;
104 }
105 } cu;

```

1.11 球面

```

1 #include <cmath>
2
3 const double PI = acos(-1.0);
4
5 //计算圆心角 lat 表示纬度, - 90 <= w <= 90, lng 表示经度
6 //返回两点所在大圆劣弧对应圆心角, 0 <= angle <= PI
7 double angle(double lng1, double lat1, double lng2, double lat2) {
8     double dlng = fabs(lng1 - lng2) * PI / 180;
9     while (dlng >= PI + PI) {
10         dlng -= PI + PI;
11     }
12     if (dlng > PI) {
13         dlng = PI + PI - dlng;
14     }
15     lat1 *= PI / 180;
16     lat2 *= PI / 180;
17     return acos(cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2));
18 }
19
20 //计算距离, r 为球半径
21 double lineDist(double r, double lng1, double lat1, double lng2, double lat2) {
22     double dlng = fabs(lng1 - lng2) * PI / 180;
23     while (dlng >= PI + PI) {
24         dlng -= PI + PI;
25     }
26     if (dlng > PI) {

```

```

27     dlng = PI + PI - dlng;
28 }
29 lat1 *= PI / 180;
30 lat2 *= PI / 180;
31 return r * sqrt(2 - 2 * (cos(lat1) * cos(lat2) * cos(dlng) + sin(lat1) * sin(lat2)))\
32 ;
33 }
34
35 //计算球面距离, r 为球半径
36 inline double sphereDist(double r, double lng1, double lat1, double lng2, double lat2) {
37     return r * angle(lng1, lat1, lng2, lat2);
38 }

```

1.12 网格 (pick)

```

1 #include <cstdlib>
2
3 struct Point {
4     int x, y;
5 };
6
7 int gcd(int a, int b) {
8     return b ? gcd(b, a % b) : a;
9 }
10
11 //多边形上的网格点个数
12 int gridOnedge(int n, const Point *p) {
13     int i, ret = 0;
14     for (i = 0; i < n; i++) {
15         ret += gcd(abs(p[i].x - p[(i + 1) % n].x), abs(p[i].y - p[(i + 1) % n].y));
16     }
17     return ret;
18 }
19
20 //多边形内的网格点个数
21 int gridInside(int n, const Point *p) {
22     int i, ret = 0;
23     for (i = 0; i < n; i++) {
24         ret += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
25     }
26     return (abs(ret) - gridOnedge(n, p)) / 2 + 1;
27 }

```

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)

$$D(n) = n!(1 - 1/1! + 1/2! - 1/3! + \dots + (-1)^n/n!) \\ = (n-1)(D(n-2) + D(n-1))$$

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

Catalan numbers:

$$Ca(n) = C(2n-2, n-1)/n$$

K-dimensional Catalan numbers:

$$A(n) = 0! * 1! * \dots * (k-1)! * (k * n)! / (n! * (n+1)! * \dots * (n+k-1)!)$$

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

```

1 //字典序全排列与序号的转换
2 int perm2Num(int n, const int *p) {
3     int ret = 0, k = 1;
4     for (int i = n - 2; i >= 0; k *= n - (i--)) {
5         for (int j = i + 1; j < n; j++) {
6             if (p[j] < p[i]) {
7                 ret += k;
8             }
9         }
10    }
11    return ret;
12 }
13
14 void num2Perm(int n, int *p, int t) {
15     for (int i = n - 1; i >= 0; i--) {
16         p[i] = t % (n - i);
17         t /= n - i;
18     }
19     for (int i = n - 1; i; i--) {
20         for (int j = i - 1; j >= 0; j--) {
21             if (p[j] <= p[i]) {
22                 p[i]++;
23             }
24         }
25     }
26 }

```

2.3 字典序组合

```

1 //字典序组合与序号的转换
2 //comb 为组合数  $C(n, m)$ , 必要时换成大数, 注意处理  $C(n, m) = 0 \mid n < m$ 
3 int comb(int n, int m) {
4     int ret = 1;
5     m = m < (n - m) ? m : (n - m);
6     for (int i = n - m + 1; i <= n; ret *= (i++));
7     for (int i = 1; i <= m; ret /= (i++));
8     return m < 0 ? 0 : ret;
9 }
10
11 int comb2Num(int n, int m, const int *c) {
12     int ret = comb(n, m);
13     for (int i = 0; i < m; i++) {
14         ret -= comb(n - c[i], m - i);
15     }
16     return ret;
17 }
18
19 void num2Comb(int n, int m, int *c, int t) {
20     int j = 1, k;
21     for (int i = 0; i < m; c[i++] = j++) {
22         for (; t > (k = comb(n - j, m - i - 1)); t -= k, j++);
23     }
24 }

```

2.4 排列组合生成

```

1 //genPerm 产生字典序排列 P(n, m)
2 //genComb 产生字典序组合 C(n, m)
3 //genPermSwap 产生相邻位对换全排列 P(n, n)
4 //产生元素用 1..n 表示
5 //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
6 const int MAXN = 100;
7
8 void dummy(const int *a, int n) {
9     //...
10 }
11
12 void genPermRc(int *a, int n, int m, int k, int *temp, bool *tag) {
13     if (k == m) {
14         dummy(temp, m);
15     } else {
16         for (int i = 0; i < n; i++) {
17             if (!tag[i]) {
18                 temp[k] = a[i], tag[i] = true;
19                 genPermRc(a, n, m, k + 1, temp, tag);
20                 tag[i] = 0;
21             }
22         }
23     }
24 }
25
26 void genPerm(int n, int m) {
27     int a[MAXN], temp[MAXN];
28     bool tag[MAXN] = {false};
29     for (int i = 0; i < n; i++) {
30         a[i] = i + 1;
31     }
32     genPermRc(a, n, m, 0, temp, tag);
33 }
34
35 void genCombRc(int *a, int s, int e, int m, int &count, int *temp) {
36     if (m == 0) {
37         dummy(temp, count);
38     } else {
39         for (int i = s; i <= e - m + 1; i++) {
40             temp[count++] = a[i];
41             genCombRc(a, i + 1, e, m - 1, count, temp);
42             count--;
43         }
44     }
45 }
46
47 void genComb(int n, int m) {
48     int a[MAXN], temp[MAXN], count = 0;
49     for (int i = 0; i < n; i++) {
50         a[i] = i + 1;
51     }
52     genCombRc(a, 0, n-1, m, count, temp);

```



```

53 }
54
55 void genPermSwapRc(int *a, int n, int k, int *pos, int *dir) {
56     int p1, p2, t;
57     if (k == n) {
58         dummy(a, n);
59     } else {
60         genPermSwapRc(a, n, k + 1, pos, dir);
61         for (int i = 0; i < k; i++) {
62             p2 = (p1 = pos[k]) + dir[k];
63             t = a[p1];
64             a[p1] = a[p2];
65             a[p2] = t;
66             pos[a[p1] - 1] = p1;
67             pos[a[p2] - 1] = p2;
68             genPermSwapRc(a, n, k + 1, pos, dir);
69         }
70         dir[k] = -dir[k];
71     }
72 }
73
74 void genPermSwap(int n) {
75     int a[MAXN], pos[MAXN], dir[MAXN];
76     for (int i = 0; i < n; i++) {
77         a[i] = i + 1;
78         pos[i] = i;
79         dir[i] = -1;
80     }
81     genPermSwapRc(a, n, 0, pos, dir);
82 }

```

2.5 生成 gray 码

```

1 //生成 reflected gray code
2 //每次调用 gray 取得下一个码
3 //000...000 是第一个码,100...000 是最后一个码
4 void gray(int n, int *code) {
5     int t = 0;
6     for (int i = 0; i < n; t += code[i++]);
7     if (t & 1) {
8         for (n--; !code[n]; n--);
9     }
10    code[n - 1] = 1 - code[n - 1];
11 }

```

2.6 置换 (polya)

```

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

```

```
5
6 int gcd(int a, int b) {
7     return b ? gcd(b, a % b) : a;
8 }
9
10 int polya(int *perm, int n, int &num) {
11     int i, j, p, v[MAXN] = {0}, ret = 1;
12     for (num = i = 0; i < n; i++) {
13         if (!v[i]) {
14             num++;
15             p = i;
16             for (j=0; !v[p = perm[p]]; j++) {
17                 v[p] = 1;
18             }
19             ret *= j / gcd(ret, j);
20         }
21     }
22     return ret;
23 }
```

Chapter 3

结构

3.1 ST 表

```
1 // RMQ
2 // MAXL = ceil(Lg(MAXN))
3 // 根据需要重写以下函数
4
5 #define BIN(i) (1 << (i))
6 #define HLF(i) (BIN(i) >> 1)
7 #define PRE(i) ((i) > 0 ? (i) - 1 : 0)
8
9 // 为了省时间可以把 lg(x) 做成表, 可以参考位运算加速
10 // 注意 lg(1 << i) = i - 1; lg((1 << i) + 1) = i;
11 const double eps = 1e-8;
12 const double ln2 = log(2.0);
13 inline int lg2(double x) {
14     return (int)floor(fabs(log(x) / ln2 - eps));
15 }
16
17 /***** 一维 *****/
18
19 // MAXL = min{(1 << MAXL) >= MAXN};
20
21 template<int MAXL, class T = int, int MAXN = 1 << MAXL>
22     struct RMQ {
23         T e[MAXN];
24         int rmq[MAXL][MAXN];
25
26         // 重写 cmp, 比较两个下标, 返回较“小”下标
27     int cmp(int l, int r) {
28         return e[l] <= e[r] ? l : r;
29     }
30
31     // 请直接对 e 赋值后调用
32 void init(const int n) {
33     for (int i = 0; i < n; i++)
34         rmq[0][i] = i;
```

```

35     for (int i = 0; BIN(i + 1) <= n; i++)
36         for (int j = 0; j <= n - BIN(i + 1); j++)
37             rmq[i + 1][j] = cmp(rmq[i][j], rmq[i][j + BIN(i)]);
38 }
39
40 // [L, r) (L < r)
41 int index(int l, int r) {
42     int b = lg2(r - l);
43     return cmp(rmq[b][l], rmq[b][r - (1 << b)]);
44 }
45 T value(int l, int r) {
46     return e[index(l, r)];
47 }
48 };
49
50
51 /***** 二维 *****/
52
53 // 如果 MLE 就把 int 改成 short
54 typedef pair<int, int> IndexType;
55
56 // MAXR = min{(1 << MAXR) >= MAXM};
57 // MAXC = min{(1 << MAXC) >= MAXN};
58
59 template<int MAXR, int MAXC, class T = int, int MAXM = 1 << MAXR, int MAXN = 1 << MAXC>
60         struct RMQ2 {
61             T e[MAXM][MAXN];
62             IndexType rmq[MAXR][MAXC][MAXM][MAXN];
63
64 IndexType cmp(const IndexType &lhs, const IndexType &rhs) {
65     return e[lhs.first][lhs.second] <= e[rhs.first][rhs.second] ? lhs : rhs;
66 }
67
68 void init(int m, int n) {
69     for (int x = 0; x < m; x++)
70         for (int y = 0; y < n; y++)
71             rmq[0][0][x][y] = make_pair(x, y);
72     for (int i = 0, ii; ii = PRE(i), BIN(i) <= m; i++)
73         for (int j = 0, jj; jj = PRE(j), BIN(j) <= n; j++)
74             for (int x = 0, xx; xx = HLF(i), x <= m - BIN(i); x++)
75                 for (int y = 0, yy; yy = HLF(j), y <= n - BIN(j); y++)
76                     rmq[i][j][x][y] = cmp(
77                         cmp(rmq[ii][jj][x]      [y], rmq[ii][jj][x]      [y + yy]),
78                         cmp(rmq[ii][jj][x + xx][y], rmq[ii][jj][x + xx][y + yy])
79                     );
80 }
81
82 IndexType index(int x1, int y1, int x2, int y2) {
83     int xx = lg2(x2 - x1), yy = lg2(y2 - y1);
84     return cmp(
85         cmp(rmq[xx][yy][x1]      [y1], rmq[xx][yy][x1]      [y2 - (1 << yy)]\
86     ),
87         cmp(rmq[xx][yy][x2 - (1 << xx)][y1], rmq[xx][yy][x2 - (1 << xx)][y2 - (1 << yy)]\
88     )
89     );

```

```

90 }
91
92 T value(int x1, int y1, int x2, int y2) {
93     IndexType i = index(x1, y1, x2, y2);
94     return e[i.first][i.second];
95 }
96
97     };
98
99  /*
100  可以在开头定义一个全局变量，代替用浮点函数的 lg2
101
102  template<int MAXN>
103  struct LG2
104  {
105
106      int lg2[MAXN + 1];
107
108      LG2()
109      {
110          lg2[0] = -1;
111          for (int i = 1; i <= MAXN; i++) {
112              lg2[i] = lg2[i - 1] + ((i & (i - 1)) == 0);
113          }
114      }
115
116      int operator()(int x) const { return lg2[x]; }
117  };
118
119  LG2<65536> lg2;
120
121  */

```

3.2 Splay

```

1 #include <algorithm>
2 using namespace std;
3
4  /*
5  每次先调用一下 Splay::init()
6  不同的题目注意修改 newNode, pushdown, update 三个函数就足够了
7  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。
8  空间吃紧时可修改 erase 函数，增加内存池管理
9  */
10 const int N = 130005;
11 struct Node {

```

```

12     int key, size;
13     bool rvs;
14     Node *f, *ch[2];
15     void set(int c, Node *x); //设当前结点的左 (0)/右 (1) 儿子为 x
16     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 \
17 明确
18     void pushdown(); //标记下传
19     void update(); //从儿子处更新自己的信息
20     void rotate(); //向上旋转
21     void Splay(Node *); //把当前 Node 旋转到参数传入的 Node 下面。Node 默认为 null, 直 \
22 接调用 Splay() 则旋转到根
23 } statePool[N], *null; //本模板统一用 null 代替 NULL
24 void Node::set(int c, Node *x) {
25     ch[c] = x;
26     x->f = this;
27 }
28 void Node::fix() {
29     ch[0]->f = this;
30     ch[1]->f = this;
31 }
32 void Node::pushdown() {
33     if (this == null) return;
34     if (rvs) {
35         ch[0]->rvs ^= 1;
36         ch[1]->rvs ^= 1;
37         rvs = 0;
38         swap(ch[0], ch[1]);
39     }
40 }
41 void Node::update() {
42     if (this == null) return;
43     size = ch[0]->size + ch[1]->size + 1;
44 }
45 void Node::rotate() {
46     Node *x = f;
47     bool o = f->ch[0] == this;
48     x->set(!o, ch[o]);
49     x->f->set(x->f->ch[1] == x, this);
50     set(o, x);
51     x->update();
52     update();
53 }
54 void Node::Splay(Node *goal = null) {
55     pushdown();
56     for (; f != goal;) {
57         f->f->pushdown();
58         f->pushdown();
59         pushdown();
60         if (f->f == goal) {
61             rotate();
62         } else if ((f->f->ch[0] == f) ^ (f->ch[0] == this)) {
63             rotate();

```

```

64         rotate();
65     } else {
66         f->rotate();
67         rotate();
68     }
69 }
70 }
71 namespace Splay {
72 int poolCnt;
73 Node *newNode() {
74     Node *p = &statePool[poolCnt++];
75     p->f = p->ch[0] = p->ch[1] = null;
76     p->size = 1;
77     p->rvs = 0;
78     return p;
79 }
80 //该命名空间里的函数必须先调用 init()
81 void init() {
82     poolCnt = 0;
83     null = newNode();
84     null->size = 0;
85 }
86
87 //用 a[l..r] 的值建立一棵完全平衡的 Splay tree。返回树根。
88 template <class T> Node *build(int l, int r, T a[]) {
89     if (l > r) return null;
90     Node *p = newNode();
91     int mid = (l + r) / 2;
92     p->key = a[mid];
93     if (l < r) {
94         p->ch[0] = build(l, mid - 1, a);
95         p->ch[1] = build(mid + 1, r, a);
96         p->update();
97         p->fix();
98     }
99     return p;
100 }
101
102 //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。
103 Node *select(Node *root, int i) {
104     for (Node *p = root;;) {
105         p->pushdown();
106         if (p->ch[0]->size + 1 == i) {
107             return p;
108         } else if (p->ch[0]->size >= i) {
109             p = p->ch[0];
110         } else {
111             i -= p->ch[0]->size + 1;
112             p = p->ch[1];
113         }
114     }
115 }
116
117 //返回结点 p 在树中的排名
118 int rank(Node *p) {

```

```

119     p->Splay();
120     return p->ch[0]->size + 1;
121 }
122
123 // 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 \
124 杂度。
125 // 返回 null 时可以这样保证复杂度: select(root, root->size)->Splay();
126 template <class T> Node *lower_bound(Node *root, T a) {
127     Node *ret = null;
128     for (Node *p = root; p != null; ) {
129         p->pushdown();
130         if (a < p->key) {
131             p = p->ch[1];
132         } else {
133             ret = p;
134             p = p->ch[0];
135         }
136     }
137     return ret;
138 }
139
140 //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。
141 Node *merge(Node *p, Node *q) {
142     p->f = q->f = null;
143     if (p == null) return q;
144     if (q == null) return p;
145     q = select(q, 1);
146     q->Splay();
147     q->set(0, p);
148     q->update();
149     return q;
150 }
151
152 //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
153 Node *reverse(Node *p, Node *q) {
154     swap(p->ch[0], q->ch[0]);
155     p->ch[1] = q->ch[1];
156     q->ch[1] = p;
157     q->f = null;
158     p->fix();
159     q->fix();
160     p->update();
161     q->update();
162     p->ch[0]->rvs ^= 1;
163     return q;
164 }
165
166 //翻转第 l 个元素到第 r 个元素, 返回树根, 下标从 1 开始。
167 Node *reverse(Node *root, int l, int r) {
168     if (l >= r) return root;
169     Node *p = select(root, l);
170     p->Splay();
171     Node *q = select(p, r);
172     q->Splay(p);

```



```

173     return reverse(p, q);
174 }
175
176 //在 p 的前一位插入 q, 返回树根
177 Node *insert(Node *p, Node *q) {
178     p->Splay();
179     if (p->ch[0] == null) {
180         p->set(0, q);
181     } else {
182         Node *t = select(p, p->ch[0]->size);
183         t->Splay(p);
184         t->set(1, q);
185         t->update();
186     }
187     p->update();
188     return p;
189 }
190
191 //在第 i 个元素前插入结点 q, 返回树根
192 Node *insert(Node *root, Node *q, int i) {
193     if (i > root->size) {
194         Node *p = select(root, root->size);
195         p->Splay();
196         p->set(1, q);
197         p->update();
198         return p;
199     } else {
200         Node *p = select(root, i);
201         return insert(p, q);
202     }
203 }
204
205 //删除以 p 为根的子树
206 Node *erase(Node *p) {
207     if (p->f != null) {
208         Node *q = p->f;
209         q->pushdown();
210         q->set(q->ch[1] == p, null);
211         q->update();
212         q->Splay();
213         return q;
214     } else {
215         return null;
216     }
217 }
218
219 //删除第 l 个到第 r 个结点, 返回树根
220 Node *erase(Node *root, int l, int r) {
221     if (l > r) return root;
222     if (l == r) {
223         Node *p = select(root, l);
224         p->Splay();
225         return merge(p->ch[0], p->ch[1]);
226     } else {
227         Node *p = select(root, l);

```

```

228     p->Splay();
229     Node *q = select(p, r);
230     q->Splay(p);
231     return merge(p->ch[0], q->ch[1]);
232 }
233 }
234
235 //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。
236 pair <Node *, Node *> split(Node *p) {
237     Node *q = p->ch[0];
238     p->ch[0] = null;
239     q->f = null;
240     p->update();
241     p->Splay();
242     return make_pair(q, p);
243 }
244 }

```

3.3 划分树

```

1 // 划分树 by yxdb
2 // 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。
3
4 // 对于  $N$  个元素的数组，空间复杂度为  $O(N\log N)$ ，初始化时间复杂度为  $O(N\log N)$ ，每次查询时间复杂度为  $O(\log N)$ 。
5
6 // 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。
7 //  $query(l, r, k)$  返回数组中  $[l, r]$  的第  $k$  小元素 ( $k$  为 1 时即为求最小值)。
8 //  $query2(l, r, x)$  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)。
9 template <class T, int N>
10 class PartitionTree {
11     pair<T,int> a[N];
12     int p[25][N], L[25][N], R[25][N];
13     bool isL[25][N];
14     int sz;
15     void build(int d, int l, int r) {
16         if (l+1==r) return;
17         int m=(l+r)>>1;
18         for (int i=l, j=l, k=m; i<r; ++i) {
19             L[d][i]=j, R[d][i]=k;
20             p[d+1][(isL[d][i]=p[d][i]<m)?j++:k++]=p[d][i];
21         }
22         build(d+1, l, m);
23         build(d+1, m, r);
24     }
25 public:
26     void build(int n, T *d) {
27         sz = n;
28         for (int i=0; i<n; ++i) a[i]=make_pair(d[i],i);
29         sort(a, a+n);
30         for (int i=0; i<n; ++i) p[0][a[i].second]=i;
31         build(0, 0, n);

```

```

32 }
33 T query(int l, int r, int k) {
34     int d, cnt;
35     for (d=0; l<r; ++d) {
36         cnt=L[d][r]-L[d][l]+isL[d][r];
37         if (cnt>=k) l=L[d][l], r=L[d][r]-!isL[d][r];
38         else k-=cnt, l=R[d][l], r=R[d][r]- isL[d][r];
39     }
40     return a[p[d][l]].first;
41     // 若改为 return a[p[d][l]].second; 可求第 k 小元素的位置.
42 }
43 int query2(int l, int r, T x) {
44     int d, cnt, ret = 0, ll = 0, rr = sz, m, tmp1 = l, tmpr = r;;
45     for (d=0; l<r; ++d) {
46         cnt=L[d][r]-L[d][l]+isL[d][r];
47         if (a[m=ll+rr>>1].first > x) l=L[d][l], r=L[d][r]-!isL[d][r], rr=m;
48         else ret += cnt, l=R[d][l], r=R[d][r]- isL[d][r], ll=m;
49     }
50     if (ret < tmpr-tmp1+1 && query(tmp1, tmpr, ret+1) < x) ret++;
51     return ret;
52 }
53 };

```

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;
}
```

```

68 void push(int x) {
69     if (f[x].rev) {
70         f[f[x].l].rev^=1;
71         f[f[x].r].rev^=1;
72         swap(f[x].l,f[x].r);
73         f[x].rev=0;
74     }
75     //
76 }
77
78 void update(int x) {
79     //
80 }
81
82 void rotate(int x) {
83     int y=f[x].par,z=f[y].par;
84     push(y);
85     push(x);
86     if (f[y].l==x) f[f[y].l=f[x].r].par=y,f[x].r=y;
87     else f[f[y].r=f[x].l].par=y,f[x].l=y;
88     if (f[z].l==y) f[z].l=x;
89     if (f[z].r==y) f[z].r=x;
90     f[f[y].par=x].par=z;
91     update(y);
92 }
93
94 void splay(int x) {
95     push(x);
96     for (; !isroot(x); rotate(x)) {
97         int y=f[x].par,z=f[y].par;
98         if (!isroot(y)) rotate(f[z].l==y?f[y].l==x?x:y);
99     }
100    update(x);
101 }
102
103 void access(int x) {
104     for(int t=0,y=x; y; y=f[y].par) splay(y),f[y].r=t,update(t=y);
105     splay(x);
106 }
107
108 void makeroot(int x) {
109     access(x);
110     f[x].rev^=1;
111 }
112
113 void link(int x,int y) {
114     makeroot(x);
115     f[x].par=y;
116 }
117
118 void cut(int x,int y) {
119     makeroot(x);
120     access(y);
121     f[x].par=f[y].l=0;
122     update(y);
123 }
124

```

```

125 int query(int x,int y) {
126     makeroot(x);
127     access(y);
128     //
129 }
130
131 void change(int x,int y) {
132     makeroot(x);
133     access(y);
134     //
135 }
136
137 bool isconnect(int x,int y) {
138     makeroot(x);
139     access(y);
140     while (push(y),f[y].l) y=f[y].l;
141     splay(y);
142     return x==y;
143 }
144
145 int main() {
146 }

```

3.5 子阵和

```

1 //求 sum{a[0..m-1][0..n-1]}
2 //维护和查询复杂度均为  $O(\log m * \log n)$ 
3 //用于动态求子阵和，数组内容保存在 Sum.a[][] 中
4 //可以改成其他数据类型
5 #include <cstring>
6
7 const int MAXN = 10000;
8
9 inline int lowbit(int x) {
10     return (x & -x);
11 }
12
13
14 template<class elemType>
15 class Sum {
16     elemType a[MAXN][MAXN], c[MAXN][MAXN], ret;
17     int m, n, t;
18     void init(int i, int j) {
19         memset(a, 0, sizeof(a));
20         memset(c, 0, sizeof(c));
21         m=i;
22         n=j;
23     }
24
25     void update(int i, int j, elemType v) {
26         for (v -= a[i][j], a[i++][j++] += v, t = j; i <= m; i += lowbit(i)) {
27             for (j = t; j <= n; j += lowbit(j)) {
28                 c[i - 1][j - 1] += v;

```

```

29     }
30 }
31 }
32
33 elemType query(int i, int j) {
34     for (ret = 0, t = j; i; i ^= lowbit(i)) {
35         for (j = t; j; j ^= lowbit(j)) {
36             ret += c[i - 1][j - 1];
37         }
38     }
39     return ret;
40 }
41 };

```

3.6 左偏树

```

1 // 左偏树是可以高效做合并操作的堆
2
3 #include <algorithm> // swap
4 #include <functional> // Less
5
6 template<typename T = int, typename Pred = less<T> >
7 struct LeftistTree {
8     struct node_type {
9         T v;
10        int d;
11        node_type *l, *r;
12        node_type(T v, int d) {
13            this->v = v;
14            this->d = d;
15            l = NULL;
16            r = NULL;
17        }
18        ~node_type() {
19            delete l;
20            delete r;
21        }
22    };
23 private:
24     node_type *root;
25     // 比较
26     static Pred pr;
27     // 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)
28     static node_type *merge(node_type *l, node_type *r) {
29         if (l == NULL) {
30             return r;
31         }
32         if (r == NULL) {
33             return l;
34         }
35
36         if (pr(r->v, l->v)) {
37             swap(l, r);

```

```

38     }
39     l->r = merge(l->r, r);
40     if (l->r != NULL && (l->l == NULL || l->r->d > l->l->d)) {
41         swap(l->l, l->r);
42     }
43     if (l->r == NULL) {
44         l->d = 0;
45     } else {
46         l->d = l->r->d + 1;
47     }
48
49     return l;
50 }
51 public:
52 LeftistTree() {
53     root = NULL;
54 }
55 ~LeftistTree() {
56     delete root;
57 }
58 // 合并操作将让被合并的树变为空
59 void merge(LeftistTree &t) {
60     root = merge(root, t.root);
61     t.root = NULL;
62 }
63 void push(T v) {
64     root = merge(root, new node_type(v, 0));
65 }
66 void pop() {
67     node_type *l = root->l, *r = root->r;
68
69     root->l = NULL;
70     root->r = NULL;
71     delete root;
72     root = merge(l, r);
73 }
74 T front() {
75     return root->v;
76 }
77 };

```

3.7 并查集

```

1 //带路径压缩的并查集，动态维护和查询等价类
2 //维护和查询复杂度略大于  $O(1)$ 
3 //集合元素取值  $0 \sim \text{MAXN}-1$ ,  $\text{find}(k)$  返回所在的并查集根的编号
4 //init 默认不等价
5 const int MAXN = 1000000;
6 struct Dset {
7     int p[MAXN];
8     void init(int N) {
9         for (int i = 0; i < N; ++i)
10             p[i] = i;
11     }

```



```

12     int find(int k) {
13         return p[k] == k ? k : (p[k] = find(p[k]));
14     }
15     void setFriend(int i, int j) {
16         p[find(i)] = p[find(j)];
17     }
18     bool isFriend(int i, int j) {
19         return find(i) == find(j);
20     }
21 };

```

3.8 扩展并查集

```

1 //扩展并查集 by asmn
2 //已知许多组  $v[i]^v[j]=k$ , 询问给某一对  $v[i]^v[j]$  的值
3 //setDiff(i,j,k) 输入  $v[i]^v[j]=k$  的关系
4 //返回值若为 false 表示输入的关系与之前有矛盾, 不予处理
5 //query(i,j,ans) 查询  $v[i]^v[j]$  的结果 ans
6 //返回若值为 false 表示结果无法由已有关系推得
7 #include <cstdio>
8 #include <algorithm>
9 using namespace std;
10 const int MAXN = 100000;
11 struct T {
12     int prv, dif;
13     T() {}
14     T(int prv, int dif):prv(prv),dif(dif) {}
15     T operator ^=(const T a) {
16         return T(prv = a.prv, dif ^= a.dif);
17     }
18 };
19 struct Dset {
20     T p[MAXN];
21     void init(int n) {
22         for (int i = 0; i < n; ++i) {
23             p[i] = T(i, 0);
24         }
25     }
26     T find(int k) {
27         return p[k].prv == k ? p[k] : (p[k] ^= find(p[k].prv));
28     }
29     // set  $v[i] ^ v[j] = k$ 
30     bool setDiff(int i, int j, int k) {
31         T ti = find(i), tj = find(j);
32         tj.dif ^= ti.dif ^ k;
33         p[ti.prv] ^= tj;
34         if (p[tj.prv].dif) {
35             p[tj.prv].dif = 0;
36             return false;
37         } else {
38             return true;
39         }

```

```

40     }
41     //query ans = v[i] ^ v[j]
42     bool query(int i, int j, int &ans) {
43         ans = find(i).dif ^ find(j).dif;
44         return (find(i).prv == find(j).prv);
45     }
46 };
47 int main() {
48     int i, j, ans;
49     char op;
50     Dset A;
51     A.init(100);
52     while (scanf("%c %d%d", &op, &i, &j) != EOF) {
53         switch (op) {
54             case 'f':
55                 if (A.setDiff(i, j, 0)) {
56                     puts("OK");
57                 } else {
58                     puts("NO");
59                 }
60                 break;
61             case 'e':
62                 if (A.setDiff(i, j, 1)) {
63                     puts("OK");
64                 } else {
65                     puts("NO");
66                 }
67                 break;
68             case 'q':
69                 if (A.query(i, j, ans)) {
70                     printf("%s\n", ans ? "e" : "f");
71                 } else {
72                     puts("I dont know");
73                 }
74             }
75         }
76     }
77 }

```

3.9 树状数组

```

1 // 树状数组 By 猛犸也钻地 @ 2011.11.24
2
3 /* 使用提示 */
4 单点修改/区间查询:
5     使 P 位置增加 V          modify(P,V);
6     查询 [L,R] 区间的和      getsum(R)-getsum(L-1);
7 区间修改/单点查询:
8     使 [L,R] 区间各增加 V    modify(L,V),modify(R+1,-V);
9     查询 P 位置的值          getsum(P);
10 区间修改/区间查询:

```

```

11 使  $[0, R]$  区间各增加  $V$      $A.modify(R, R*V), B.modify(R, -V);$ 
12 查询  $[0, R]$  区间的和     $A.getsum(R) + B.getsum(R) * R;$ 
13 多维树状数组和单维的类似, 比如修改操作是这样的:
14  $for(int\ i=x+BIAS; i<SIZE; i++)$ 
15  $for(int\ j=y+BIAS; j<SIZE; j++)\ u[i][j] += v;$ 
16 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
17 // 因此  $SIZE$  通常设为最大结点数  $+2*BIAS$ , 而  $BIAS$  通常设为  $5$  或  $10$  */
18
19 #include <cstring>
20 using namespace std;
21
22 class BITree {
23 public:
24     static const int SIZE = 100010, BIAS = 5;
25     long long u[SIZE];
26     void clear() {
27         memset(u, 0, sizeof(u));
28     }
29     void modify(int x, long long v) {
30         for(x+=BIAS; x<SIZE; x+=x&-x) u[x] += v;
31     }
32     long long getsum(int x) {
33         long long s = 0;
34         for(x+=BIAS; x; x-=x&-x) s += u[x];
35         return s;
36     }
37 };

```

3.10 树链剖分

```

1  // 树链剖分 By 猛犸也钻地 @ 2012.02.10
2
3  #include <vector>
4  #include <cstring>
5  using namespace std;
6
7  class TreeDiv {
8  public:
9      static const int SIZE = 100005; // SIZE 为最大结点数 +1
10     int sz[SIZE], lv[SIZE]; // sz[] 为子树的总结点数, lv[] 为结点的深度
11     int rt[SIZE], fa[SIZE]; // rt[] 为结点所在的树根, fa[] 为结点的父亲
12     // seg[] 为结点所在的链的编号, idx[] 为结点在链上的位置
13     // low[] 为链首结点, top[] 为链尾结点的父亲, len[] 为链上结点的个数
14     int cnt, seg[SIZE], idx[SIZE], low[SIZE], top[SIZE], len[SIZE];
15     // 遍历某条链的方法: for(int x=low[u]; x!=top[u]; x=fa[x])
16     // 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
17     int init(int n, const vector<int> e[]) {
18         memset(len, 0, sizeof(len));

```

```

19     memset(lv,0,sizeof(lv));
20     for(int i=0; i<n; i++) if(!lv[i]) {
21         segment(rt[i]=fa[i]=i,e);
22         top[seg[i]]=fa[i]=SIZE-1; // 指向虚根
23     }
24     return cnt;
25 }
26 // 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)
27 int lca(int x, int y) {
28     if(rt[x]!=rt[y]) return -1;
29     while(seg[x]!=seg[y]) {
30         int p=top[seg[x]],q=top[seg[y]];
31         if(lv[p]>lv[q]) x=p;
32         else y=q;
33     }
34     return lv[x]<lv[y]?x:y;
35 }
36 private:
37 void segment(int x, const vector<int> e[]) {
38     int y,t=-1;
39     sz[x]=1;
40     rt[x]=rt[fa[x]];
41     lv[x]=lv[fa[x]]+1;
42     for(size_t i=0; i<e[x].size(); i++) if(e[x][i]!=fa[x]) {
43         fa[y=e[x][i]]=x;
44         segment(y,e);
45         sz[x]+=sz[y];
46         if(t<0 || sz[y]>sz[t]) t=y;
47     }
48     seg[x]=~t?seg[t]:cnt;
49     idx[x]=~t?idx[t]+1:0;
50     if(t<0) low[cnt++]=x;
51     len[seg[x]]++;
52     top[seg[x]]=fa[x];
53 }
54 };

```

3.11 矩形并

```

1 //线段树求矩形并得面积和周长
2 //要保证传入的 x1 < x2, y1 < y2
3 #include<algorithm>
4 #include <cstring>
5 #define MAXN 40000
6 using namespace std;
7 typedef long long LL;
8 const int K = 1;
9 int tot, X[MAXN * 2], CX;
10 pair<pair<int, int>, pair<int,int> > E[MAXN * 2];
11
12 //矩形类
13 struct Rect {
14     int x1, y1, x2, y2;

```

```

15 } R[MAXN];
16
17 struct SegTree {
18     int L, R, Lson, Rson, cover, length[K + 1];
19     void build(int, int);
20     void add(int, int, int);
21 } T, A[MAXN * 4];
22
23 void SegTree::build(int l, int r) {
24     L = l;
25     R = r;
26     cover = 0;
27     memset(length, 0, sizeof(length));
28     length[0] = X[R + 1] - X[L];
29     if (L == R) return;
30     int mid = (L + R) >> 1;
31     A[Lson = tot++].build(l, mid);
32     A[Rson = tot++].build(mid + 1, r);
33 }
34 //线段树的填删线段, 只能删除之前加入过的线段
35 void SegTree::add(int v, int l, int r) {
36     memset(length, 0, sizeof(length));
37     if (l <= L && R <= r) {
38         cover += v;
39         if (L == R) {
40             length[min(cover, K)] += X[R + 1] - X[L];
41         } else {
42             for (int i = 0; i <= K; ++i) {
43                 length[min(i + cover, K)] += A[Lson].length[i] + A[Rson].length[i];
44             }
45         }
46         return;
47     }
48     int mid = (L + R) >> 1;
49     if (l <= mid) A[Lson].add(v, l, r);
50     if (r > mid) A[Rson].add(v, l, r);
51     for (int i = 0; i <= K; ++i) {
52         length[min(i + cover, K)] += A[Lson].length[i] + A[Rson].length[i];
53     }
54 }
55
56 //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
57 void discrete(Rect *R, int N) {
58     int i, tx1, tx2;
59     for (CX = i = 0; i < N; ++i) {
60         X[CX++] = R[i].x1;
61         X[CX++] = R[i].x2;
62     }
63     sort(X, X + CX);
64     CX = unique(X, X + CX) - X;
65     T.build(tot = 0, CX - 2);
66     for (i = 0; i < N; ++i) {
67         tx1 = lower_bound(X, X + CX, R[i].x1) - X;
68         tx2 = lower_bound(X, X + CX, R[i].x2) - X;
69         E[i].second = E[i + N].second = make_pair(tx1, tx2);
70         E[i].first = make_pair(R[i].y1, -1);
71         E[i + N].first = make_pair(R[i].y2, 1);

```

```

72     }
73     sort(E, E + (N << 1));
74 }
75
76 //求矩形并得周长, 传入储存矩形的数组 R 和矩形个数 N
77 int perimeter(Rect *R, int N) {
78     int ret = 0, i, k, prv;
79     for (k = 0; k < 2; ++k) {
80         discrete(R, N);
81         for (prv = i = 0; i < (N << 1); ++i) {
82             T.add(-E[i].first.second, E[i].second.first, E[i].second.second - 1);
83             ret += abs(T.length[K] - prv);
84             prv = T.length[K];
85         }
86         for (i = 0; i < N; ++i) {
87             swap(R[i].x1, R[i].y1);
88             swap(R[i].x2, R[i].y2);
89         }
90     }
91     return ret;
92 }
93
94 //求矩形并的面积, 传入储存矩形的数组 R 和矩形个数 N
95 LL area(Rect *R, int N) {
96     int i, k, prv;
97     LL ret = 0;
98     discrete(R, N);
99     prv = E[0].first.first;
100    for(i = 0; i < (N << 1); ++i) {
101        ret += (LL)T.length[K] * (E[i].first.first - prv);
102        prv = E[i].first.first;
103        T.add(-E[i].first.second, E[i].second.first, E[i].second.second - 1);
104    }
105    return ret;
106 }
107 int main() {}

```

3.12 线段树

```

1 //线段树
2 //可以处理加入边和删除边不同的情况
3 //incSet 和 decSeg 用于加入边
4 //segLen 求长度
5 //t 传根节点 (一律为 1)
6 //L0, r0 传树的节点范围 (一律为 1..t)
7 //L, r 传线段 (端点)
8 const int MAXN = 10000;
9 class SegTree {
10     int n, cnt[MAXN], len[MAXN];
11
12     SegTree(int t) : n(t) {

```

```

13     for (int i = 1; i <= t; i++) {
14         cnt[i] = len[i] = 0;
15     }
16 };
17
18 void update(int t, int L, int r);
19 void incSet(int t, int L0, int r0, int L, int r);
20 void decSeg(int t, int L0, int r0, int L, int r);
21 int segLen(int t, int L0, int r0, int L, int r);
22 };
23
24 inline int length(int L, int r) {
25     return r - L;
26 }
27
28 void SegTree::update(int t, int L, int r) {
29     if (cnt[t] || r - L == 1) {
30         len[t] = length(L, r);
31     } else {
32         len[t] = len[t + t] + len[t + t + 1];
33     }
34 }
35
36 void SegTree::incSet(int t, int L0, int r0, int L, int r) {
37     if (L0 == L && r0 == r) {
38         cnt[t]++;
39     } else {
40         int m0 = (L0 + r0) >> 1;
41         if (L < m0) {
42             incSet(t + t, L0, m0, L, m0 < r ? m0 : r);
43         }
44         if (r > m0) {
45             incSet(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
46         }
47         if (cnt[t + t] && cnt[t + t + 1]) {
48             cnt[t + t]--;
49             update(t + t, L0, m0);
50             cnt[t + t + 1]--;
51             update(t + t + 1, m0, r0);
52             cnt[t]++;
53         }
54     }
55     update(t, L0, r0);
56 }
57
58 void SegTree::decSeg(int t, int L0, int r0, int L, int r) {
59     if (L0 == L && r0 == r) {
60         cnt[t]--;
61     } else if (cnt[t]) {
62         cnt[t]--;
63         if (L > L0) {
64             incSet(t, L0, r0, L0, L);
65         }
66         if (r < r0) {
67             incSet(t, L0, r0, r, r0);
68         }
69     } else {
70         int m0 = (L0 + r0) >> 1;

```

```

71         if (L < m0) {
72             decSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
73         }
74         if (r > m0) {
75             decSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
76         }
77     }
78     update(t, L0, r0);
79 }
80
81 int SegTree::segLen(int t, int L0, int r0, int L, int r) {
82     if (cnt[t] || (L0 == L && r0 == r)) {
83         return len[t];
84     } else {
85         int m0 = (L0 + r0) >> 1, ret = 0;
86         if (L < m0) {
87             ret += segLen(t + t, L0, m0, L, m0 < r ? m0 : r);
88         }
89         if (r > m0) {
90             ret += segLen(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
91         }
92         return ret;
93     }
94 }

```

3.13 线段树扩展

```

1 //线段树扩展
2 //可以计算长度和线段数
3 //可以处理加入边和删除边不同的情况
4 //incSeg 和 decSeg 用于加入边
5 //segLen 求长度, setCut 求线段数
6 //t 传根节点 (一律为 1)
7 //L0, r0 传树的节点范围 (一律为 1..t)
8 //L, r 传线段 (端点)
9 const int MAXN = 10000;
10 class SegTree {
11 public:
12     int n, cnt[MAXN], len[MAXN], cut[MAXN], bl[MAXN], br[MAXN];
13
14     SegTree(int t) : n(t) {
15         for (int i = 1; i <= t; i++) {
16             cnt[i] = len[i] = cut[i] = bl[i] = br[i] = 0;
17         }
18     };
19
20     void update(int t, int L, int r);
21     void incSeg(int t, int L0, int r0, int L, int r);
22     void decSeg(int t, int L0, int r0, int L, int r);
23     int segLen(int t, int L0, int r0, int L, int r);
24     int setCut(int t, int L0, int r0, int L, int r);
25 };

```



```

26
27 inline int length(int L, int r) {
28     return r - L;
29 }
30
31 void SegTree::update(int t, int L, int r) {
32     if (cnt[t] || r - L == 1) {
33         len[t] = length(L, r);
34         cut[t] = bl[t] = br[t] = 1;
35     } else {
36         len[t] = len[t + t] + len[t + t + 1];
37         cut[t] = cut[t + t] + cut[t + t + 1];
38         if (br[t + t] && bl[t + t + 1]) {
39             cut[t]--;
40         }
41         bl[t] = bl[t + t];
42         br[t] = br[t + t + 1];
43     }
44 }
45
46 void SegTree::incSeg(int t, int L0, int r0, int L, int r) {
47     if (L0 == L && r0 == r) {
48         cnt[t]++;
49     } else {
50         int m0 = (L0 + r0) >> 1;
51         if (L < m0) {
52             incSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
53         }
54         if (r > m0) {
55             incSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
56         }
57         if (cnt[t + t] && cnt[t + t + 1]) {
58             cnt[t + t]--;
59             update(t + t, L0, m0);
60             cnt[t + t + 1]--;
61             update(t + t + 1, m0, r0);
62             cnt[t]++;
63         }
64     }
65     update(t, L0, r0);
66 }
67
68 void SegTree::decSeg(int t, int L0, int r0, int L, int r) {
69     if (L0 == L && r0 == r) {
70         cnt[t]--;
71     } else if (cnt[t]) {
72         cnt[t]--;
73         if (L > L0) {
74             incSeg(t, L0, r0, L0, L);
75         }
76         if (r < r0) {
77             incSeg(t, L0, r0, r, r0);
78         }
79     } else {
80         int m0 = (L0 + r0) >> 1;
81         if (L < m0) {
82             decSeg(t + t, L0, m0, L, m0 < r ? m0 : r);
83         }

```

```

84         if (r > m0) {
85             decSeg(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
86         }
87     }
88     update(t, L0, r0);
89 }
90
91 int SegTree::segLen(int t, int L0, int r0, int L, int r) {
92     if (cnt[t] || (L0 == L && r0 == r)) {
93         return len[t];
94     } else {
95         int m0 = (L0 + r0) >> 1, ret=0;
96         if (L < m0) {
97             ret += segLen(t + t, L0, m0, L, m0 < r ? m0 : r);
98         }
99         if (r > m0) {
100             ret += segLen(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
101         }
102         return ret;
103     }
104 }
105
106 int SegTree::setCut(int t, int L0, int r0, int L, int r) {
107     if (cnt[t]) {
108         return 1;
109     }
110     if (L0 == L && r0 == r) {
111         return cut[t];
112     } else {
113         int m0 = (L0 + r0) >> 1, ret = 0;
114         if (L < m0) {
115             ret += setCut(t + t, L0, m0, L, m0 < r ? m0 : r);
116         }
117         if (r > m0) {
118             ret += setCut(t + t + 1, m0, r0, m0 > L ? m0 : L, r);
119         }
120         if (L < m0 && r > m0 && br[t + t] && bl[t + t + 1]) {
121             ret--;
122         }
123         return ret;
124     }
125 }

```

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

```
1 //分解质因数
2 //primeFactor() 传入 n, 返回不同质因数的个数
3 //f 存放质因数, nf 存放对应质因数的个数
4 //先调用 initPrime(), 其中第二个 initPrime() 更快
5
6 #include <cmath>
7
8 const int PSIZE = 100000;
9
10 int plist[PSIZE], pcount = 0;
11
12 bool prime(int n) {
```

```

13     if ((n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n % 5)) || (n != 7 \
14 && !(n % 7))) {
15         return false;
16     }
17     for (int i = 0; plist[i] * plist[i] <= n; ++i) {
18         if (n % plist[i] == 0) {
19             return false;
20         }
21     }
22     return n > 1;
23 }
24
25 void initPrime() {
26     plist[pcount++] = 2;
27     for (int i = 3; i < 1000000; i += 2) {
28         if (prime(i)) {
29             plist[pcount++] = i;
30         }
31     }
32 }
33
34 int primeFactor(int n, int *f, int *nf) {
35     int cnt = 0;
36     int n2 = (int) sqrt((double) n);
37     for (int i = 0; n > 1 && plist[i] <= n2; ++i) {
38         if (n % plist[i] == 0) {
39             for (nf[cnt] = 0; n % plist[i] == 0; n /= plist[i]) {
40                 ++nf[cnt];
41             }
42             f[cnt++] = plist[i];
43         }
44     }
45     if (n > 1) {
46         nf[cnt] = 1;
47         f[cnt++] = n;
48     }
49     return cnt;
50 }
51
52 //产生 MAXN 以内的所有素数
53 //note:2863311530 就是 101010101010101010101010101010
54 //给所有 2 的倍数赋初值
55 #include <cmath>
56
57 const int MAXN = 100000000;
58 unsigned int plist[6000000], pcount;
59 unsigned int isPrime[(MAXN >> 5) + 1];
60
61 #define setbitzero(a) (isPrime[(a) >> 5] &= (~(1 << ((a) & 31))))
62 #define setbitone(a) (isPrime[(a) >> 5] |= (1 << ((a) & 31)))
63 #define ISPRIME(a) (isPrime[(a) >> 5] & (1 << ((a) & 31)))
64
65 void initPrime() {
66     int i, j, m;
67     int t = (MAXN >> 5) + 1;
68     for (i = 0; i < t; ++i) {

```

```

69     isPrime[i] = 2863311530;
70 }
71 plist[0] = 2;
72 setbitone(2);
73 setbitzero(1);
74 m = (int) sqrt((double) MAXN);
75 pcount = 1;
76 for (i = 3; i <= m; i += 2) {
77     if (ISPRIME(i)) {
78         plist[pcount++] = i;
79         for (j = i << 1; j <= MAXN; j += i) {
80             setbitzero(j);
81         }
82     }
83 }
84 if (!(i & 1)) {
85     ++i;
86 }
87 for (; i <= MAXN; i += 2) {
88     if (ISPRIME(i)) {
89         plist[pcount++] = i;
90     }
91 }
92 }
93
94 // O(n) 筛素数表
95 // 返回素数个数
96 // 素数保存在 plist 里面
97 // minp 存放最小的素因子
98 #include <cstring>
99
100 const int MAXN = 1000000;
101
102 int minp[MAXN + 1], plist[MAXN + 1];
103
104 int prime(int n = MAXN) {
105     int num = 0;
106     memset(minp, 0, sizeof(minp));
107     for (int i = 2; i <= n; i++) {
108         if (!minp[i]) plist[num++] = i, minp[i] = i;
109         for (int j = 0; j < num && i * plist[j] <= n; j++) {
110             minp[i * plist[j]] = plist[j];
111             if (i % plist[j] == 0) break;
112         }
113     }
114     return num;
115 }
116
117 // 先调用 prime 初始化
118 // 然后就可以调用 factorize 分解质因素
119 // 结果存在 p 里面, 返回质因数个数
120 // 要保证 n >= 2
121 int factorize(int n, int *p) {

```

```

122     int num = 0;
123     while (n != 1) {
124         p[num++] = minp[n];
125         n /= minp[n];
126     }
127     return num;
128 }

```

4.3 同余方程合并

```

1  /*
2     同余方程合并
3      $x \% m[0] = c[0]$ 
4      $x \% m[1] = c[1]$ 
5     ...
6      $x \% m[n-1] = c[n-1]$ 
7      $0 \leq c[i] < m[i], \text{LCM}(m[i]) < LL$ 
8     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
9     可以处理  $m[i]$  不互素的情况,  $c[i] \geq m[i]$  将认为无解
10  */
11  typedef long long ll;
12
13  ll exgcd(ll a, ll b, ll &x, ll &y) {
14      if (!b)
15          return x = 1, y = 0, a;
16      ll d = exgcd(b, a % b, x, y), t = x;
17      y = t - (a / b) * (x = y);
18      return d;
19  }
20
21  inline ll mod(ll x, ll y) { //  $y > 0$ 
22      return (x %= y) < 0 ? x + y : x;
23  }
24
25  ll solve(int n, ll m[], ll c[]) {
26      for (int i = 0; i < n; i++)
27          if (c[i] >= m[i]) return -1;
28      ll ans = c[0], LCM = m[0], x, y, g;
29      for (int i = 1; i < n; i++) {
30          if (LCM < m[i]) // 防止溢出, 如能保证  $m[i]$  在 int 范围内可以删去
31              swap(LCM, m[i]), swap(ans, c[i]);
32          g = exgcd(LCM, m[i], x, y);
33          if ((c[i] - ans) % g) return -1;
34          ans += LCM * mod((c[i] - ans) / g * x, m[i] / g);
35          ans %= LCM * m[i] / g;
36      }
37      return ans;
38  }

```

4.4 数论变换

```

1 #include<algorithm>
2 using namespace std;
3 typedef long long LL;
4 LL pm(LL a,LL n,LL m) {
5     LL r=1;
6     for(; n; n>>=1,a=a*a%m)
7         if(n&1)r=r*a%m;
8     return r;
9 }
10 //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
11 //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,
12 //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
13 //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
14 //就要用 NTT, 几个可替换的 (P,g) 对:
15 //(2113929217,5),(1811939329,13),(2130706433,3)
16 const LL P=2013265921,g=31;
17 void ntt(LL *a,size_t n,bool inv=false) {
18     // inv 为 true 时表示逆运算;
19     LL w=1,d=pm(g,(P-1)/n,P),t;
20     size_t i,j,c,s;
21     if(inv) {
22         for(i=1,j=n-1; i<j; swap(a[i++],a[j--]));
23         for(t=pm(n,P-2,P),i=0; i<n; ++i)a[i]=a[i]*t%P;
24     }
25     for(s=n>>1; s; s>>=1,d=d*d%P)
26         for(c=0; c<s; ++c,w=w*d%P)
27             for(i=c; i<n; i+=s<<1) {
28                 a[i|s]=(a[i|s]+P-(t=a[i|s]))*w%P;
29                 a[i]=(a[i]+t)%P;
30             }
31     for(i=1; i<n; ++i) {
32         for(j=0,s=i,c=n>>1; c; c>>=1,s>>=1)j=j<<1|s&1;
33         if(i<j)swap(a[i],a[j]);
34     }
35 }
36 // 计算卷积姿势,
37 // 比如计算 a[0..n-1] 和 b[0..m-1] 的卷积:c[0..n+m-1]
38 // int l=2;
39 // for(;l<n+m-1;l<=1);
40 // ntt(a,l);ntt(b,l);
41 // for(int i=0;i<l;++i)c[i]=a[i]*b[i]%P;
42 // ntt(c,l,true);

```

4.5 模线性方程 (组)

```

1 //扩展 Euclid 求解  $\gcd(a,b)=ax+by$ 
2 int extGcd(int a, int b, int &x, int &y) {
3     int t, ret;
4     if (!b) {
5         x = 1;
6         y = 0;
7         return a;
8     }
9     ret = extGcd(b, a % b, x, y);
10    t = x;
11    x = y;
12    y = t - a / b * y;
13    return ret;
14 }
15
16 //计算  $m^a$ ,  $O(\log a)$ , 本身没什么用, 注意这个按位处理的方法 :-P
17 int exponent(int m, int a) {
18     int ret = 1;
19     for (; a >>= 1, m *= m) {
20         if (a & 1) {
21             ret *= m;
22         }
23     }
24     return ret;
25 }
26
27 //计算幂取模  $a^b \bmod n$ ,  $O(\log b)$ 
28 int modularExponent(int a, int b, int n) {
29     //  $a^b \bmod n$ 
30     int ret = 1;
31     for (; b >>= 1, a = (int) ((long long) a * a % n)) {
32         if (b & 1) {
33             ret = (int) ((long long) ret * a % n);
34         }
35     }
36     return ret;
37 }
38
39 //求解模线性方程  $ax=b \pmod n$ 
40 //返回解的个数, 解保存在 sol[] 中
41 //要求  $n>0$ , 解的范围  $0..n-1$ 
42 int modularLinear(int a, int b, int n, int *sol) {
43     int d, e, x, y, i;
44     d = extGcd(a, n, x, y);
45     if (b % d) {
46         return 0;
47     }
48     e = (x * (b / d) % n + n) % n;
49     for (i = 0; i < d; i++) {
50         sol[i] = (e + i * (n / d)) % n;
51     }

```



```

52     return d;
53 }
54
55 //求解模线性方程组 (中国余数定理)
56 // x = b[0] (mod w[0])
57 // x = b[1] (mod w[1])
58 // ...
59 // x = b[k-1] (mod w[k-1])
60 //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]
61 int modularLinearSystem(int b[], int w[], int k) {
62     int d, x, y, a = 0, m, n = 1, i;
63     for (i = 0; i < k; i++) {
64         n *= w[i];
65     }
66     for (i = 0; i < k; i++) {
67         m = n / w[i];
68         d = extGcd(w[i], m, x, y);
69         a = (a + y * m * b[i]) % n;
70     }
71     return (a + n) % n;
72 }

```

4.6 欧拉函数

```

1 int gcd(int a, int b) {
2     return b ? gcd(b, a % b) : a;
3 }
4
5 inline int lcm(int a, int b) {
6     return a / gcd(a, b) * b;
7 }
8
9 //求 1..n-1 中与 n 互质的数的个数
10 int eular(int n) {
11     int ret = 1, i;
12     for (i = 2; i * i <= n; i++) {
13         if (n % i == 0) {
14             n /= i;
15             ret *= i - 1;
16             while (n % i == 0) {
17                 n /= i;
18                 ret *= i;
19             }
20         }
21     }
22     if (n > 1) {
23         ret *= n - 1;
24     }
25     return ret;
26 }
27
28 // O(n) 求 1 到 n 的欧拉函数, 顺便筛出一个素数表

```

```

29 // 素数保存在 plist 里面, euler 存放欧拉函数值
30 // 返回素数个数
31 #include <cstring>
32
33 const int MAXN = 1000000;
34
35 int euler[MAXN + 1], plist[MAXN + 1];
36
37 int doEuler(int n = MAXN) {
38     int num = 0;
39     memset(euler, 0, sizeof(euler));
40     euler[1] = 1;
41     for (int i = 2; i <= n; i++) {
42         if (!euler[i]) plist[num++] = i, euler[i] = i - 1;
43         for (int j = 0; j < num && i * plist[j] <= n; j++) {
44             if (i % plist[j] == 0)
45                 euler[i * plist[j]] = euler[i] * plist[j];
46             else
47                 euler[i * plist[j]] = euler[i] * (plist[j] - 1);
48             if (i % plist[j] == 0) break;
49         }
50     }
51     return num;
52 }

```

4.7 离散对数及原根

```

1  /*
2      离散对数
3      传入 a,b,p, 返回  $a^x \bmod p = b$  的最小解 x, 若无解返回 -1.
4      可以处理 p 不为素数的情况
5      代码中把  $b \geq p$  的情况判为无解
6      算法复杂度  $O(p^{0.5} \lg(p^{0.5}))$ 
7  */
8  int gcd(int a, int b) {
9      return b ? gcd(b, a % b) : a;
10 }
11
12 int pow_mod(int a, long long b, int p) {
13     int r = 1;
14     if (p == 1) return 0;
15     for (; b >= 1; a = (long long)a * a % p)
16         if (b & 1) r = (long long)r * a % p;
17     return r;
18 }
19
20 int phi(int n) {
21     int i, m = n;
22     for (i = 2; n / i >= i; i++) if (n % i == 0) {
23         m = m / i * (i - 1);
24         while (n % i == 0) n /= i;
25     }

```

```

26     if (n>1) m=m/n*(n-1);
27     return m;
28 }
29
30 map <int,int> Mp;
31
32 int logorithm(int a,int b,int p) {
33     if (b>=p) return -1;
34     Mp.clear();
35     int r=0,d,i,j,t1,t2,m=(int)ceil(sqrt(p)+1e-9),m1;
36     if (p==1) return 0;
37     for(i=0,t1=1; i<32; i++) {
38         if (t1==b) return i;
39         t1=(long long)t1*a%p;
40     }
41     for(t1=d=1; gcd((long long)t1*a%p,p)!=d; r++) {
42         t1=(long long)t1*a%p;
43         d=gcd(t1,p);
44     }
45     if (b%d!=0) return -1;
46     if (t1==b) return r;
47     Mp[t1]=0;
48     int pre=t1;
49     for(i=1; i<m; i++) {
50         int tmp=1LL*pre*a%p;
51         if (!Mp.count(tmp)) Mp[tmp]=i;
52         pre=tmp;
53         if (tmp==b) return r+i;
54     }
55     m1=phi(p/d);
56     m1=pow_mod(a,m1-m%m1,p);
57     for(i=1; i<m; i++) {
58         b=(long long)b*m1%p;
59         map <int,int>::iterator it=Mp.find(b);
60         if (it!=Mp.end()) return i*m+it->second+r;
61     }
62     return -1;
63 }
64
65 // O(n^0.5) 进行质因数分解
66 // cnt 为不同质因数个数, fac 为所有不同质因数
67 void primeFactor(int n, int &cnt, int fac[]) {
68     cnt = 0;
69     for (int i = 2; i * i <= n; ++i)
70         if (n % i == 0)
71             for (fac[cnt++] = i; n % i == 0; n /= i);
72     if (n > 1)
73         fac[cnt++] = n;
74 }
75
76 // O(n^0.5+gdLogn) 求素数最小原根 g
77 int primitiveRootForPrime(int n) {
78     static int cnt, fac[30];
79     if (n == 2) return 1;
80     primeFactor(n - 1, cnt, fac);
81     for (int j, i = 2; i < n; ++i) {

```

```

82         for (j = 0; j < cnt && pow_mod(i, (n - 1) / fac[j], n) != 1; ++j);
83         if (j >= cnt) return i;
84     }
85     return 0;
86 }
87
88 // O(n^0.5+gdLogn) 求任意正整数最小原根 g, 无原根返回 0
89 int primitiveRoot(int n) {
90     static int cnt, fac[30];
91     if (n == 2) return 1;
92     if (n == 4) return 3;
93     if (n % 4 == 0) return 0;
94     primeFactor(n, cnt, fac);
95     if (cnt > 2 || cnt == 2 && fac[0] != 2) return 0;
96     int p1 = fac[0], p2 = cnt == 2 ? fac[1] : p1;
97     int phi = ::phi(n);
98     primeFactor(phi, cnt, fac);
99     for (int j, i = 2; i < n; ++i) {
100         if (i % p1 == 0 || i % p2 == 0) continue;
101         for (j = 0; j < cnt && pow_mod(i, phi / fac[j], n) != 1; ++j);
102         if (j >= cnt) return i;
103     }
104     return 0;
105 }

```

4.8 简易素数表

```

1 //用素数表判定素数, 先调用 initPrime
2 int plist[10000], pcount = 0;
3
4 bool prime(int n) {
5     int i;
6     if ((n != 2 && !(n % 2)) || (n != 3 && !(n % 3)) || (n != 5 && !(n % 5)) || (n != 7 \
7 && !(n % 7))) {
8         return false;
9     }
10    for (i = 0; plist[i] * plist[i] <= n; i++) {
11        if (!(n % plist[i])) {
12            return false;
13        }
14    }
15    return n > 1;
16 }
17
18 void initPrime() {
19     plist[pcount++] = 2;
20     for (int i = 3; i < 50000; i += 2) {
21         if (prime(i)) {
22             plist[pcount++] = i;
23         }
24     }
25 }

```

4.9 阶乘最后非零位

```

1 //求阶乘最后非零位, 复杂度  $O(n\log n)$ 
2 //返回该位,  $n$  以字符串方式传入
3 #include <cstring>
4 const int MAXN = 10000;
5
6 int lastdigit(char *buf) {
7     const int mod[20] = {
8         1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2
9     };
10    int len = strlen(buf), a[MAXN], i, c, ret = 1;
11    if (len == 1) {
12        return mod[buf[0] - '0'];
13    }
14    for (i = 0; i < len; i++) {
15        a[i] = buf[len - 1 - i] - '0';
16    }
17    for (; len; len -= !a[len - 1]) {
18        ret = ret * mod[a[1] % 2 * 10 + a[0]] % 5;
19        for (c = 0, i = len - 1; i >= 0; i--) {
20            c = c * 10 + a[i];
21            a[i] = c / 5;
22            c %= 5;
23        }
24    }
25    return ret + ret % 2 * 5;
26 }

```

4.10 高级素数表

```

1 // 质数的判定/筛选/分解 By 猛犸也钻地 @ 2012.02.21
2
3 #include <vector>
4 #include <cmath>
5 #include <algorithm>
6 using namespace std;
7
8 class FastSieve {
9 public:
10    static const int MOD = 1000000007, SIZE = 1000050;
11    typedef long long LL;
12    vector<int> pl, lo, eu, rv;
13    // 线性筛出  $[0, SIZE)$  范围内的质数
14    //  $lo[]$  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
15    FastSieve() {
16        lo=eu=rv=vector<int>(SIZE);
17        lo[1]=eu[1]=rv[1]=1;
18        for(int x=2; x<SIZE; x++) {
19            rv[x]=rv[MOD%x]*LL(MOD-MOD/x)%MOD;
20            if(!lo[x]) pl.push_back(lo[x]=x), eu[x]=x-1;
21            for(size_t i=0; i<pl.size() && x*pl[i]<SIZE; i++) {
22                lo[x*pl[i]]=pl[i];

```

```

23         eu[x*pl[i]]=eu[x]*(pl[i]-(x%pl[i]!=0));
24         if(x%pl[i]==0) break;
25     }
26 }
27 }
28 // 对 n 做质因数分解
29 vector<LL> factorize(LL n) {
30     vector<LL> u;
31     int i,t=sqrt(n+1);
32     for(i=0; pl[i]<=t; i++) if(n%pl[i]==0) {
33         do {
34             n/=pl[i];
35             u.push_back(pl[i]);
36         } while(n%pl[i]==0);
37         t=sqrt(n+1);
38     }
39     if(n>1) u.push_back(n);
40     return u;
41 }
42 // 判断 n 是否为质数
43 bool prime(LL n) {
44     if(n<SIZE) return n>=2 && lo[n]==n;
45     int i,t=sqrt(n+1);
46     for(i=0; pl[i]<=t; i++) if(n%pl[i]==0) return false;
47     return true;
48 }
49 };
50
51 class Primality {
52 public:
53     typedef long long LL;
54     // 判断 num 是否为质数
55     bool miller_rabin(LL num) {
56         if(num<=1) return false;
57         if(num<=3) return true;
58         if(~num&1) return false;
59         const int u[] = {2,325,9375,28178,450775,9780504,1795265022,0};
60         //const int u[]={2,3,5,7,11,13,17,19,23,29,0};
61         LL e=num-1,a,c=0;
62         while(~e&1) e/=2,c++;
63         for(int i=0; u[i]; i++) {
64             if(num<=u[i]) return true;
65             a=POW(u[i],e,num);
66             if(a==1) continue;
67             for(int j=1; a!=num-1; j++) {
68                 if(j==c) return false;
69                 a=MUL(a,a,num);
70             }
71         }
72         return true;
73     }
74     // 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
75     LL pollard_rho(LL n) {
76         if(n<=3 || miller_rabin(n)) return n; // 保证 n 为合数时可去掉这行

```

```

77     while(1) {
78         int i=1,cnt=2;
79         LL x=rand()%n,y=x,c=rand()%n;
80         if(!c || c==n-2) c++;
81         do {
82             LL u=__gcd(n-x+y,n);
83             if(u>1 && u<n) return u;
84             if(++i==cnt) y=x,cnt*=2;
85             x=(c+MUL(x,x,n))%n;
86         } while(x!=y);
87     }
88     return n;
89 }
90 // 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
91 vector<LL> factorize(LL n) {
92     vector<LL> u;
93     if(n>1) u.push_back(n);
94     for(size_t i=0; i<u.size(); i++) {
95         LL x=pollard_rho(u[i]);
96         if(x==u[i]) continue;
97         u[i--]/=x;
98         u.push_back(x);
99     }
100     sort(u.begin(),u.end());
101     return u;
102 }
103 // 返回 x 与 y 相乘模 m 的结果, x 与 m 要小于 2^62
104 LL MUL(LL a, LL b, LL mod) { //O(1) Long Long 乘法
105     assert(0 <= a && a < mod);
106     assert(0 <= b && b < mod);
107     if (mod < int(1e9)) return a * b % mod;
108     LL k = (LL)((long double)a * b / mod);
109     LL res = a * b - k * mod;
110     res %= mod;
111     if (res < 0) res += mod;
112     return res;
113 }
114 /* 普通的快速乘
115 LL MUL(LL x, LL y, LL m){
116     LL c,s=0;
117     for(c=x%m;y;c=(c+c)%m,y>>=1)
118         if(y&1) s=(s+c)%m;
119     return s;
120 }*/
121 // 返回 x 的 y 次方模 m 的结果, x 与 m 要小于 2^62
122 LL POW(LL x, LL y, LL m) {
123     LL c,s=1;
124     for(c=x%m; y; c=MUL(c,c,m),y>>=1)
125         if(y&1) s=MUL(s,c,m);
126     return s;
127 }

```

128 | };

数值

```

1 //用傅立叶变换可在  $O(n\log n)$  求卷积
2 //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
3 //常数较大
4 #include <complex>
5 void FFT(complex<double> *a, int len, int inv) { //eps=1e-12
6     for (int i=0, n1=0, n2=0; i < len; ++i, n2 ^= (len/(i&-i)>>1), n1^=(i&-i))
7         if (n1 > n2)
8             swap(a[n1], a[n2]);
9     for(int m = 1; m <= len >> 1; m <= 1) {
10         complex<double> w0(cos(PI / m), sin(PI / (inv * m))), w = 1, t;
11         for(int k = 0; k < len; k += (m <= 1), w = 1)
12             for(int j = 0; j < m; ++j, w *= w0) {
13                 t=w*a[k+j+m];
14                 a[k+j+m]=a[k+j]-t;
15                 a[k+j]+=t;
16             }
17     }
18     if(inv == -1)
19         for(int i = 0; i < len; ++i)
20             a[i] /= len;
21 }

```

```

1  /* 追赶法解周期性方程，复杂度  $O(n)$ 
2     周期性方程定义：
3     
$$\begin{cases} b_0 x_0 + c_0 = x_0 \\ a_1 x_0 + b_1 x_1 + c_1 = x_1 \\ \vdots \\ a_{n-2} x_{n-2} + b_{n-2} x_{n-1} + c_{n-2} = x_{n-1} \\ a_{n-1} x_{n-1} + b_{n-1} x_n + c_{n-1} = x_n \end{cases}$$

4     输入：a[], b[], c[], x[], n
5     输出：求解结果 x 在 x[] 中

```

```

9  */
10
11 void run(double a[], double b[], double c[], double x[], int n) {
12     c[0] /= b[0];
13     a[0] /= b[0];
14     x[0] /= b[0];
15     for (int i = 1; i < n - 1; i++) {
16         double temp = b[i] - a[i] * c[i - 1];
17         c[i] /= temp;
18         x[i] = (x[i] - a[i] * x[i - 1]) / temp;
19         a[i] = -a[i] * a[i - 1] / temp;
20     }
21     a[n - 2] = -a[n - 2] - c[n - 2];
22     for (int i = n - 3; i >= 0; i--) {
23         a[i] = -a[i] - c[i] * a[i + 1];
24         x[i] -= c[i] * x[i + 1];
25     }
26     x[n - 1] -= (c[n - 1] * x[0] + a[n - 1] * x[n - 2]);
27     x[n - 1] /= (c[n - 1] * a[0] + a[n - 1] * a[n - 2] + b[n - 1]);
28     for (int i = n - 2; i >= 0; i--) {
29         x[i] += a[i] * x[n - 1];
30     }
31 }

```

5.3 多项式求根 (牛顿法)

```

1  /* 牛顿法解多项式的根
2  输入：多项式系数 c[], 多项式度数 n, 求在 [a,b] 间的根
3  输出：根
4  要求保证 [a,b] 间有根
5  */
6  #include <cmath>
7  #include <cstdlib>
8
9  double f(int m, double c[], double x) {
10     int i;
11     double p = c[m];
12     for (i = m; i > 0; i--) {
13         p = p * x + c[i - 1];
14     }
15     return p;
16 }
17
18 int newton(double x0, double *r, double c[], double cp[], int n, double a, double b, double eps) {
19     const int MAX_ITERATION = 1000;
20     int i = 1;
21     double x1, x2, fp, eps2 = eps / 10.0;
22     x1 = x0;
23     while (i < MAX_ITERATION) {
24         x2 = f(n, c, x1);
25         fp = f(n - 1, cp, x1);
26         if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0)) {
27

```

```

28         return 0;
29     }
30     x2 = x1 - x2 / fp;
31     if (fabs(x1 - x2) < eps2) {
32         if (x2 < a || x2 > b) {
33             return 0;
34         }
35         *r = x2;
36         return 1;
37     }
38     x1 = x2;
39     i++;
40 }
41 return 0;
42 }
43
44 double polynomialRoot(double c[], int n, double a, double b, double eps) {
45     double *cp;
46     int i;
47     double root;
48
49     cp = (double *) calloc(n, sizeof(double));
50     for (i = n - 1; i >= 0; i--) {
51         cp[i] = (i + 1) * c[i + 1];
52     }
53     if (a > b) {
54         root = a;
55         a = b;
56         b = root;
57     }
58     if ((!newton(a, &root, c, cp, n, a, b, eps)) && (!newton(b, &root, c, cp, n, a, b, e\
59 ps))) {
60         newton((a + b) * 0.5, &root, c, cp, n, a, b, eps);
61     }
62     free(cp);
63     if (fabs(root) < eps) {
64         return fabs(root);
65     } else {
66         return root;
67     }
68 }

```

5.4 定积分计算 (Romberg)

```

1  /* Romberg 求定积分
2
3  输入：积分区间 [a,b], 被积函数 f(x,y,z)
4
5  输出：积分结果
6
7  f(x,y,z) 示例：
8
9  double f0( double x, double L, double t )
10 {
11     return sqrt(1.0+L*(t*t*cos(t*x)*cos(t*x)));

```

```

9     }
10    */
11    #include <cmath>
12
13    double romberg(double a, double b, double(*f)(double x, double y, double z), double eps,\
14    double l, double t) {
15        const int MAXN = 1000;
16        int i, j, temp2, min;
17        double h, R[2][MAXN], temp4;
18
19        for (i = 0; i < MAXN; i++) {
20            R[0][i] = 0.0;
21            R[1][i] = 0.0;
22        }
23        h = b - a;
24        min = (int)(log(h * 10.0) / log(2.0)); //h should be at most 0.1
25        R[0][0] = ((*f)(a, l, t) + (*f)(b, l, t)) * h * 0.50;
26        i = 1;
27        temp2 = 1;
28        while (i < MAXN) {
29            i++;
30            R[1][0] = 0.0;
31            for (j = 1; j <= temp2; j++) {
32                R[1][0] += (*f)(a + h * ((double)j - 0.50), l, t);
33            }
34            R[1][0] = (R[0][0] + h * R[1][0]) * 0.50;
35            temp4 = 4.0;
36            for (j = 1; j < i; j++) {
37                R[1][j] = R[1][j - 1] + (R[1][j - 1] - R[0][j - 1]) / (temp4 - 1.0);
38                temp4 *= 4.0;
39            }
40            if ((fabs(R[1][i - 1] - R[0][i - 2]) < eps) && (i > min)) {
41                return R[1][i - 1];
42            }
43            h *= 0.50;
44            temp2 *= 2;
45            for (j = 0; j < i; j++) {
46                R[0][j] = R[1][j];
47            }
48        }
49        return R[1][MAXN - 1];
50    }
51
52    double integral(double a, double b, double(*f)(double x, double y, double z), double eps\
53    , double l, double t) {
54        const double PI = 3.1415926535897932;
55        int n;
56        double R, p, res;
57
58        n = (int)(floor)(b * t * 0.50 / PI);
59        p = 2.0 * PI / t;
60        res = b - (double)n * p;
61        if (n) {
62            R = romberg(a, p, f, eps / (double)n, l, t);
63        }
64        R = R * (double)n + romberg(0.0, res, f, eps, l, t);

```

```

65
66     return R / 100.0;
67 }
68
69 // 其实不妨先考虑用复化 Simpson 公式
70 //  $S = h / 6 * [f(A) + 4 * \sum f(X_{k+1/2}) + 2 * \sum f(X_k) + f(B)]$ ;  $k = 0..n-1$ 

```

5.5 定积分计算 (变步长 simpson)

```

1 //变步长辛普森积分
2 double func(double x) {
3     return ...; //根据题目需要实现
4 }
5 double Simpson_VariStep(double x1,double xh,double eps) {
6     int subs=1,n=1,i;
7     double result,x,p,width=xh-x1,t1=width*(func(x1)+func(xh))/2.0,t2;
8     double s1=t1,s2=s1+2.0*eps;
9     while(subs) {
10         for (p=0.0,i=0; i<=n-1; ++i) {
11             x=x1+(i+0.5)*width;
12             p=p+func(x);
13         }
14         t2=(t1+width*p)/2.0;
15         s2=(t2*4-t1)/3.0;
16         result=s2;
17         subs=(fabs(s2-s1)>=eps);
18         t1=t2;
19         s1=s2;
20         n+=n;
21         width=width/2.0;
22     }
23     return s2;
24 }

```

5.6 定积分计算 (自适应 simpson)

```

1 #include <cmath>
2 using namespace std;
3 typedef double flt;
4 //用于无法确定如何分割区间使得积分精确的情况。
5 flt f(flt x) {
6     return x; //被积函数, 根据需要进行实现
7 }
8 flt adaptive(flt l,flt r,flt m,flt eps,flt s,flt fl,flt fr,flt fm,int lim) {
9     flt h=(r-l)/12;
10    flt u=ldexp(l+m,-1),v=ldexp(m+r,-1);
11    flt fu=f(u),fv=f(v);
12    flt s1=(fl+fm+ldexp(fu,2))*h,
13        sr=(fm+fr+ldexp(fv,2))*h,
14        s2=s1+sr;
15    if(lim<=0 || fabs(s2-s)<= eps)

```

```

16         return s2+(s2-s)/15;
17     eps=ldexp(eps,-1);
18     --lim;
19     return adaptive(l,m,u,eps,sl,fl,fm,fu,lim)
20         + adaptive(m,r,v,eps,sr,fr,fv,lim);
21 }
22 // eps 是需要精度, maxrec 是限制最大迭代次数.
23 // 积分区间 [L,r]
24 flt simpson(flt l,flt r,flt eps=1e-6,int maxrec=16) {
25     flt m=ldexp(l+r,-1);
26     flt fl=f(l),fm=f(m),fr=f(r);
27     return adaptive(l,r,m,eps*15,(r-l)/6*(fl+fr+ldexp(fm,2)),fl,fr,fm,maxrec);
28 }

```

5.7 线性相关

```

1 //判线性相关 (正交化)
2 //传入 m 个 n 维向量
3 #include <cmath>
4 const int MAXN = 100;
5 const double EPS = 1e-10;
6
7 bool linearDependent(int m, int n, double vec[][MAXN]) {
8     double ort[MAXN][MAXN], e;
9     int i, j, k;
10    if (m > n) {
11        return true;
12    }
13    for (i = 0; i < m; i++) {
14        for (j = 0; j < n; j++) {
15            ort[i][j] = vec[i][j];
16        }
17        for (k = 0; k < i; k++) {
18            for (e = j = 0; j < n; j++) {
19                e += ort[i][j] * ort[k][j];
20            }
21            for (j = 0; j < n; j++) {
22                ort[i][j] -= e * ort[k][j];
23            }
24            for (e = j = 0; j < n; j++) {
25                e += ort[i][j] * ort[i][j];
26            }
27            if (fabs(e = sqrt(e)) < EPS) {
28                return 1;
29            }
30            for (j = 0; j < n; j++) {
31                ort[i][j] /= e;
32            }
33        }
34    }
35    return false;
36 }

```

5.8 线性规划

```

1 #include<algorithm>
2 #include<cmath>
3 #include<cstring>
4 using namespace std;
5 typedef double dbl;
6 const dbl inf = 1e99, eps = 1e-8;
7 //注意此 LP 处理的情况都是  $x_i$  是非负的结果
8 //如果  $x_i$  是无限限制, 则用  $x_i = x_j - x_k$  去变换
9 //如果  $x_i$  是小于某个数  $V$ , 则用  $x_i = V - x_j$  去变换
10 struct LP {
11     static const size_t N=20, M=20, L=N+M*2, faux=~0;
12     dbl A[M][L], b[M], c[N+M], c1[L], x[N], d[L], zm;
13     size_t bs[M], n, m, l;
14 #define go(i,n) for(i=0;i<n;++i)
15     dbl simplex(dbl *c) {
16         for(size_t j,k,i,o;; bs[o]=i) {
17             dbl dmax=0, u=inf, t;
18             i=o=-1;
19             go(k,l) {
20                 d[k]=c[k];
21                 go(j,m) d[k]-=c[bs[j]]*A[j][k];
22                 if(d[k]>dmax+eps) {
23                     dmax=d[k];
24                     i=k;
25                 }
26             }
27             if(!~i) {
28                 zm=0;
29                 fill(x,x+n,0);
30                 go(j,m) if(bs[j]<n)
31                     zm+=c[bs[j]]*(x[bs[j]]=b[j]);
32                 return zm;
33             }
34             go(j,m) if(A[j][i]>eps&&b[j]/A[j][i]+eps<u)
35                 u=b[o=j]/A[j][i];
36             if(!~o) throw "unbounded!"; //可行域无界, 无最大值
37             b[o]*=t=1/A[o][i];
38             go(k,l) A[o][k]*=t;
39             go(j,m) if(j!=o&&fabs(A[j][i])>eps) {
40                 t=-A[j][i];
41                 b[j]+=t*b[o];
42                 go(k,l) A[j][k]+=t*A[o][k];
43             }
44         }
45     }
46     void init(int _n) {
47         memset(A,0,sizeof(A)); //方程左边系数
48         memset(b,0,sizeof(b)); //方程右边的值
49         memset(c,0,sizeof(c)); //目标函数系数
50         l=n=_n;
51         m=0;

```

```

52     }
53     void add_constraint(dbl *a,dbl _b,bool eq=false) {
54         // eq 为 false: 添加约束为不等式  $a_0*x_0+a_1*x_1+\dots \leq _b$ 
55         // eq 为 true: 添加约束为等式  $a_0*x_0+a_1*x_1+\dots = _b$ 
56         copy(a,a+n,A[m]);
57         b[m]=_b;
58         if(eq||_b<0)bs[m]=faux;
59         else A[m][bs[m]=1++]=1;
60         m++;
61     }
62     dbl solve() {
63         size_t ol=1,j,k;
64         go(j,m)if(bs[j]==faux)A[j][bs[j]=1++]=1;
65         if(ol!=1) {
66             go(k,1)c1[k]=k<ol?0:-1;
67             if(simplex(c1)<-eps)throw "infeasible"; //无可行解
68             l=ol;
69         }
70         return simplex(c);
71     }
72 };

```

5.9 高斯消元 (全主元)

```

1  #include <cmath>
2
3  const int MAXN = 100;
4  const double EPS = 1e-10;
5
6  //全主元 gauss 消去解  $a[i][j]x[j]=b[i]$ 
7  //返回是否有唯一解, 若有解在  $b[i]$  中
8  bool gaussTpivot(int n, double a[][MAXN], double b[]) {
9      int i, j, k, row, col, index[MAXN];
10     double maxp, t;
11     for (i = 0; i < n; i++) {
12         index[i] = i;
13     }
14     for (k = 0; k < n; k++) {
15         for (maxp = 0, i = k; i < n; i++) {
16             for (j = k; j < n; j++) {
17                 if (fabs(a[i][j]) > fabs(maxp)) {
18                     maxp = a[row = i][col = j];
19                 }
20             }
21         }
22         if (fabs(maxp) < EPS) {
23             return false;
24         }
25         if (col != k) {
26             for (i = 0; i < n; i++) {
27                 swap(a[i][col], a[i][k]);
28             }
29             swap(index[col], index[k]);

```



```

30     }
31     if (row != k) {
32         for (j = k; j < n; j++) {
33             swap(a[k][j], a[row][j]);
34         }
35         swap(b[k], b[row]);
36     }
37     for (j = k + 1; j < n; j++) {
38         a[k][j] /= maxp;
39         for (i = k + 1; i < n; i++) {
40             a[i][j] -= a[i][k] * a[k][j];
41         }
42     }
43     b[k] /= maxp;
44     for (i = k + 1; i < n; i++) {
45         b[i] -= b[k] * a[i][k];
46     }
47 }
48 for (i = n - 1; i >= 0; i--) {
49     for (j = i + 1; j < n; j++) {
50         b[i] -= a[i][j] * b[j];
51     }
52 }
53 for (k = 0; k < n; k++) {
54     a[0][index[k]] = b[k];
55 }
56 for (k = 0; k < n; k++) {
57     b[k] = a[0][k];
58 }
59 return true;
60 }

```

5.10 高斯消元 (列主元)

```

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

```

```

19         if(k!=r)for(j=c; j<=m; ++j)swap(e[k][j],e[r][j]);
20         if(fabs(e[r][c])<eps)continue;
21         pos[r]=c;
22         for(i=r+1; i<n; ++i) {
23             flt t=-e[i][c]/e[r][c];
24             for(int j=c; j<=m; ++j)e[i][j]+=t*e[r][j];
25         }
26         ++r; //注意, 前面有 continue, 不能写到 for 里面去
27     }
28     if(r>0 && pos[r-1]==m)return -1;
29     for(int i=0; i<m; ++i)x[i]=0;
30     for(int i=r-1; i>=0; --i) {
31         int c=pos[i];
32         x[c]=e[i][m]/e[i][c];
33         for(int j=0; j<i; ++j)e[j][m]-=e[j][c]*x[c];
34     }
35     return r;
36 }
37 };

```

Chapter 6

字符串

6.1 Trie 图 (dd engi)

```
1  /*
2      Trie 图，即用于多串匹配的字符串自动机。
3      对于字符串  $S$  中是否存在模式串  $s_1, s_2, \dots, s_n$  的匹配的问题，
4      可以用  $O((|s_1| + |s_2| + \dots + |s_n|) / |\Sigma|)$  的时间预处理， $O(|S|)$  的时间回答询问。
5      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。
6      注意事项：
7          1. 字符的类型、字符集的大小可改。
8          2.  $ELEMENT\_MAX$  一般可设为  $|s_1| + |s_2| + \dots + |s_n|$  的最大可能值。
9          3. 传递的字符串中每个字符应在  $[0, SIGMA)$  的区间内。
10     使用方法：
11         1. 处理每个测试点前调用 init() 初始化。
12         2. 用 insert 插入  $s_1, s_2, \dots$ 
13         3. 调用 build_graph 建立 trie 图。
14         4. 调用 match 查询  $S$  中是否存在匹配。
15  */
16 #include <cstring>
17 #include <queue>
18 #include <cstdio>
19 using namespace std;
20 typedef struct node *trie;
21 const int SIGMA = 26;
22 const int ELEMENT_MAX = 50000;
23 int tot;
24 struct node {
25     bool match;
26     trie pre, child[SIGMA];
27 } T[ELEMENT_MAX];
28
29 void trie_init() {
30     tot = 1;
31     memset(T, 0, sizeof(T));
```

```

32 }
33 void insert(char *s, int n) {
34     trie t = T;
35     for (int i = 0; i < n; ++i) {
36         int c = s[i] - 'a';
37         if (!t->child[c]) {
38             t->child[c] = &T[tot++];
39         }
40         t = t->child[c];
41     }
42     t->match = true;
43 }
44 void build_graph() {
45     trie t = T;
46     queue <trie> Q;
47     for (int i = 0; i < SIGMA; ++i) {
48         if (t->child[i]) {
49             t->child[i]->pre = t;
50             Q.push(t->child[i]);
51         } else {
52             t->child[i] = t;
53         }
54     }
55     while (!Q.empty()) {
56         t = Q.front();
57         Q.pop();
58         t->match |= t->pre->match;
59         for (int i = 0; i < SIGMA; ++i) {
60             if (t->child[i]) {
61                 t->child[i]->pre = t->pre->child[i];
62                 Q.push(t->child[i]);
63             } else {
64                 t->child[i] = t->pre->child[i];
65             }
66         }
67     }
68 }
69 bool match(char *s, int n) {
70     trie t = T;
71     for (int i = 0; i < n; ++i) {
72         int c = s[i] - 'a';
73         t = t->child[c];
74     }
75     return t->match;
76 }
77 //示例程序
78 int main() {
79     trie_init();
80     insert("abcd", 4);
81     insert("bc", 2);
82     build_graph();
83
84     printf("%s\n", match("abc", 3) ? "Yes" : "No"); //output: Yes
85 }

```

6.2 Trie 图 (猛犸也钻地)

```

1 // 确定性 AC 自动机 (Trie 图) By 猛犸也钻地 @ 2011.11.24
2
3 #include <cstring>
4 #include <algorithm>
5 using namespace std;
6
7 class TrieGraph {
8 public:
9     static const int SIZE = 100005; // 最大结点总数, 约为模板串长度之和
10    static const int LEAF = 26; // 每个结点下的叶子数量
11    // 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
12    int next[SIZE], e[SIZE][LEAF], n; // e[][] 为结点的各个叶子的编号
13    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
14    TrieGraph() {
15        n=SIZE; // 别忘了写上这行
16    }
17    void init() {
18        fill_n(next, n, 0);
19        fill_n(data, n, 0);
20        memset(e, -1, n*sizeof(e[0]));
21        n=1;
22    }
23    void insert(const char *s, int idx = 0) {
24        int x=0;
25        for(int i=0; s[i]; i++) {
26            int c=s[i]-'a'; // 根据题目的字符集修改这里的映射方式
27            x=~e[x][c]?e[x][c]:e[x][c]=n++;
28        }
29        data[x]|=1<<idx;
30    }
31    void make() {
32        static int q[SIZE], m;
33        next[0]=m=0;
34        for(int c=0; c<LEAF; c++)
35            if(~e[0][c]) next[q[m++]]=e[0][c]=0;
36            else e[0][c]=0;
37        for(int i=0; i<m; i++) {
38            int x=q[i];
39            data[x]=data[next[x]]; // 求 next[] 路径上的前缀和
40            for(int c=0; c<LEAF; c++) {
41                int t=e[next[x]][c];
42                if(~e[x][c]) next[q[m++]]=e[x][c]=t;
43                else e[x][c]=t;
44            }
45        }
46    }
47 };

```

6.3 后缀数组 -线性

```

1 namespace SA {
2 int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
3 #define pushS(x) sa[cur[s[x]]--] = x
4 #define pushL(x) sa[cur[s[x]]++] = x
5 #define inducedSort(v) fill_n(sa, n, -1); fill_n(cnt, m, 0);
6     for (int i = 0; i < n; i++) cnt[s[i]]++;
7     for (int i = 1; i < m; i++) cnt[i] += cnt[i-1];
8     for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;
9     for (int i = n1-1; ~i; i--) pushS(v[i]);
10    for (int i = 1; i < m; i++) cur[i] = cnt[i-1];
11    for (int i = 0; i < n; i++) if (sa[i] > 0 && t[sa[i]-1]) pushL(sa[i]-1);
12    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;
13    for (int i = n-1; ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
14 void sais(int n, int m, int *s, int *t, int *p) {
15     int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
16     for (int i = n-2; ~i; i--) t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
17     for (int i = 1; i < n; i++) rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
18     inducedSort(p);
19     for (int i = 0, x, y; i < n; i++) if (~(x = rk[sa[i]])) {
20         if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ch++;
21         else for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++)
22             if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {
23                 ch++;
24                 break;
25             }
26         s1[y = x] = ch;
27     }
28     if (ch+1 < n1) sais(n1, ch+1, s1, t+n, p+n1);
29     else for (int i = 0; i < n1; i++) sa[s1[i]] = i;
30     for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
31     inducedSort(s1);
32 }
33 template<typename T>
34 int mapCharToInt(int n, const T *str) {
35     int m = *max_element(str, str+n);
36     fill_n(rk, m+1, 0);
37     for (int i = 0; i < n; i++) rk[str[i]] = 1;
38     for (int i = 0; i < m; i++) rk[i+1] += rk[i];
39     for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
40     return rk[m];
41 }
42 // Ensure that str[n] is the unique lexicographically smallest character in str.
43 template<typename T>
44 void suffixArray(int n, const T *str) {
45     int m = mapCharToInt(++n, str);
46     sais(n, m, s, t, p);
47     for (int i = 0; i < n; i++) rk[sa[i]] = i;
48     for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
49         int j = sa[rk[i]-1];
50         while (i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
51         if (ht[rk[i]] = h) h--;
52     }
53 }
54 };

```

6.4 后缀数组

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // partial_sum 在头文件 numeric 中
4 // 数组下标后后缀都从 0 开始标号, 空串不作考虑
5 // sa[i] 表示第 i 小的后缀从字符串的第 sa[i] 个位置开始
6 // 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
7 // ht[i] 表示 sa[i] 和 sa[i-1] 所代表字符串的公共前缀的长度
8 struct SuffixArray {
9 public:
10     const static int MAXN = 100000 + 10;
11     int cnt[MAXN], tr[2][MAXN], ts[MAXN];
12     int sa[MAXN], rk[MAXN], ht[MAXN], len; //len 字符串长度
13     // 对字符串 s[0..n-1] 做后缀排序, s[n] 最好为 '\0'
14     // 排序完, sa[0..n-1] 为有效后缀, 空串不作考虑
15     void construct(const char *s, int n, int m=256) {
16         int i,j,k,*x=tr[0],*y=tr[1];
17         this->len = n;
18         memset(cnt,0,sizeof(cnt[0])*m);
19         for (i=0; i<n; ++i) cnt[s[i]]++;
20         partial_sum(cnt,cnt+m,cnt);
21         for (i=0; i<n; ++i) rk[i]=cnt[s[i]]-1;
22         for (k=1; k<=n; k<=1) {
23             for (i=0; i<n; ++i) x[i]=rk[i],y[i]=i+k<n?rk[i+k]+1:0;
24             fill(cnt,cnt+n+1,0);
25             for (i=0; i<n; ++i) cnt[y[i]]++;
26             partial_sum(cnt,cnt+n+1,cnt);
27             for (i=n-1; i>=0; --i) ts[--cnt[y[i]]]=i;
28             fill(cnt,cnt+n+1,0);
29             for (i=0; i<n; ++i) cnt[x[i]]++;
30             partial_sum(cnt,cnt+n+1,cnt);
31             for (i=n-1; i>=0; --i) sa[--cnt[x[ts[i]]]]=ts[i];
32             for (i=rk[sa[0]]=0; i+1<n; ++i) {
33                 rk[sa[i+1]]=rk[sa[i]]+(x[sa[i]]!=x[sa[i+1]]||y[sa[i]]!=y[sa[i+1]]);
34             }
35         }
36         for (i=0,k=0; i<n; ++i) {
37             if (!rk[i]) continue;
38             for (j=sa[rk[i]-1]; i+k<n&&j+k<n&&s[i+k]==s[j+k];) k++;
39             ht[rk[i]]=k;
40             if (k) k--;
41         }
42         rmq_init(n);
43     }
44     // 求任意两个后缀的 Lcp
45     inline int lcp(int a, int b) {
46         a=rk[a],b=rk[b];
47         if (a == b) return len - a + 1;
48         if (a>b) swap(a,b);
49         return rmq(a+1,b);
50     }

```

```

51 private:
52     int mx[MAXN][20], LOG[MAXN];
53     void rmq_init(int n) {
54         for (int i=-(LOG[0]=-1); i<n; ++i) LOG[i]=LOG[i>>1]+1;
55         for (int i=0; i<n; ++i) mx[i][0]=ht[i];
56         for (int i,j=1; (1<<j)<n; ++j) {
57             for (i=0; i+(1<<j)<=n; ++i)
58                 mx[i][j]=min(mx[i][j-1],mx[i+(1<<(j-1))][j-1]);
59         }
60     }
61     inline int rmq(int a, int b) {
62         int k=LOG[b-a+1];
63         return min(mx[a][k],mx[b-(1<<k)+1][k]);
64     }
65 };

```

6.5 后缀自动机

```

1  /*
2  这是求解多串 LCS 的例程
3  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)
4  每次 add 至多加入两个点
5  大致来说建成的后缀自动机有这么几个性质
6  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 \
7  上面跑, 不可能遇到 null 边
8  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。
9  3、val 表示的是能从 root 转移到这个状态的最长串的长度
10 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]
115、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。
126、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 \
13比 fa 的要小
14 */
15
16 template <class T> void checkmin(T &t,T x) {
17     if (x < t) t = x;
18 }
19 template <class T> void checkmax(T &t,T x) {
20     if (x > t) t = x;
21 }
22 #define foreach(it,v) for (__typeof((v).begin()) it = (v).begin();it != (v).end();it++)
23 const int N = 250005;
24
25 struct Node {
26     Node *ch[26], *fa;
27     int val;
28     int len[10];
29     Node():
30         val(0), fa(NULL) {
31             memset(ch, 0, sizeof(ch));

```



```

32     memset(len, 0, sizeof(len));
33 }
34 } pool[N * 2 + 5], *last, *root;
35 vector <Node *> vec[N];
36
37 namespace SAM {
38 int cnt;
39
40 void init() {
41     if (cnt)
42         for (int i = 0; i < cnt; i++)
43             pool[i] = Node();
44     cnt = 1;
45     root = &pool[0];
46     last = root;
47 }
48
49 void add(int c) {
50     Node *p = last, *np = &pool[cnt++];
51     last = np;
52     np->val = p->val + 1;
53     for (; p && !p->ch[c]; p = p->fa)
54         p->ch[c] = np;
55     if (!p) {
56         np->fa = root;
57     } else {
58         Node *q = p->ch[c];
59         if (p->val + 1 == q->val) {
60             np->fa = q;
61         } else {
62             Node *nq = &pool[cnt++];
63             *nq = *q;
64             nq->val = p->val + 1;
65             q->fa = nq;
66             np->fa = nq;
67             for (; p && p->ch[c] == q; p = p->fa)
68                 p->ch[c] = nq;
69         }
70     }
71 }
72 }
73
74 int m, n;
75 char S[N], T[N];
76
77 int main() {
78     SAM::init();
79     scanf("%s", S);
80     m = strlen(S);
81     for (int i = 0; i < m; i++)
82         SAM::add(S[i] - 'a');
83     int k;
84     for (k = 0; scanf("%s", T) != EOF; k++) {
85         Node *p = root;
86         int cnt = 0;
87         n = strlen(T);
88         for (int i = 0; i < n; i++) {
89             int c = T[i] - 'a';

```

```

90         if (p->ch[c]) {
91             cnt++;
92             p = p->ch[c];
93         } else {
94             for (; p && !p->ch[c]; p = p->fa);
95             if (!p) {
96                 p = root;
97                 cnt = 0;
98             } else {
99                 cnt = p->val + 1;
100                 p = p->ch[c];
101             }
102         }
103         checkmax(p->len[k], cnt);
104     }
105 }
106 for (int i = 0; i < SAM::cnt; i++) {
107     vec[pool[i].val].push_back(&pool[i]);
108 }
109 int ans = 0;
110 for (int i = m; i >= 0; i--) {
111     foreach (it, vec[i]) {
112         Node *p = *it;
113         int now = p->val;
114         for (int j = 0; j < k; j++) {
115             checkmin(now, p->len[j]);
116             if (p->fa) {
117                 checkmax(p->fa->len[j], p->len[j]);
118             }
119         }
120         checkmax(ans, now);
121     }
122 }
123 printf("%d\n", ans);
124 }

```

6.6 回文树

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 // palindromic tree, O(N)
4 // 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
5 // 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
6 // 首先调用 PT::init() 传入字符串 p 和字符串长度 n
7 // 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
8 // 添加完成之后, 调用 PT::count(), 计算每个回文串出现次数
9 // 功能:
10 // 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
11 // 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
12 // 3. 统计出现次数最多的回文串, max(pool[i].cnt)

```

```

13 // 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
14 namespace PT {
15 static const int MAXN = 300000 + 10, SIGMA = 26;
16 struct Node {
17     Node *ch[SIGMA], *fa;
18     int len, num, cnt;
19 } pool[MAXN], *last; //last 当前串的最长回文后缀
20 int sz;
21 char *s;
22 void init(char p[], int n) {
23     memset(pool, 0, sizeof(pool[0]) * (n + 10));
24     s = p;
25     last = &pool[1];
26     sz = 2;
27     pool[0].len = -1;
28     pool[0].fa = &pool[0];
29     pool[1].len = 0;
30     pool[1].fa = &pool[0];
31 }
32 bool add(int pos) {
33     Node *cur = last;
34     int curlen = 0, c = s[pos] - 'a';
35     while (1) {
36         curlen = cur->len;
37         if (pos >= 1 + curlen && s[pos - 1 - curlen] == s[pos]) break;
38         cur = cur->fa;
39     }
40     if (cur->ch[c]) return last = cur->ch[c], last->cnt ++, false;
41     last = &pool[sz ++];
42     last->cnt ++;
43     last->len = cur->len + 2;
44     cur->ch[c] = last;
45     if (last->len == 1) return last->fa = &pool[1], last->num = 1;
46     while (1) {
47         cur = cur->fa;
48         curlen = cur->len;
49         if (pos >= curlen + 1 && s[pos - 1 - curlen] == s[pos]) {
50             last->fa = cur->ch[c];
51             break;
52         }
53     }
54     return last->num = last->fa->num + 1;
55 }
56 void count() {
57     for (int i = sz - 1; i > 1; -- i) pool[i].fa->cnt += pool[i].cnt;
58 }
59 }
60
61 const int MAXN = 300000 + 10;
62 char s[MAXN];
63
64 int main() {
65     scanf("%s", s);
66     int len = strlen(s);
67     PT::init(s, len);
68     // total distinct palindromic string of s is <= len

```

```

69 // new add a letter will add 0 or 1 new palindromic string
70 for (int i = 0; i < len; ++ i) {
71     PT::add(i); // true: add a new palindromic string
72     // cout << PT::last->num; // number of palindrome ends with s[i]
73     // cout << PT::sz - 2 << endl; // number of distinct palindrome in prefix [0..i-1]
74     1]
75     // cout << ret << endl; // total palindrome in prefix [0..i-1]
76 }
77 PT::count(); // calc occurrence of each palindrome
78 // max(PT::pool[i].cnt * PT::pool[i].Len)
79 return 0;
80 }

```

6.7 字符串最小表示

```

1  /*
2   求字符串的最小表示
3   输入：字符串
4   返回：字符串最小表示的首字母位置 (0...size-1)
5  */
6  #include <vector>
7  using namespace std;
8
9  template <class T>
10 int minString(const vector<T> &str) {
11     int i, j, k;
12     vector<T> ss(str.size() << 1);
13     for (i = 0; i < str.size(); i++) {
14         ss[i] = ss[i + str.size()] = str[i];
15     }
16     for (i = k = 0, j = 1; k < str.size() && i < str.size() && j < str.size(); ) {
17         for (k = 0; k < str.size() && ss[i + k] == ss[j + k]; k++);
18         if (k < str.size()) {
19             if (ss[i + k] > ss[j + k]) {
20                 i += k + 1;
21             } else {
22                 j += k + 1;
23             }
24             if (i == j) {
25                 j++;
26             }
27         }
28     }
29     return i < j ? i : j;
30 }

```

6.8 最长回文子串

```

1 // 最长回文子串 (Manacher) By 猛犸也钻地 @ 2012.11.29
2
3 #include <vector>
4 #include <algorithm>
5 using namespace std;
6
7 // 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)
8 int manacher(const char *s, int n) {
9     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
10    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
11        u[i]=max(x+u[x]-i,1-i%2); // 相应位置的直径长度就是 10301410121
12        if(x+x>=i) u[i]=min(u[i],u[x+x-i]);
13        int a=(i-1-u[i])>>1,b=(i+1+u[i])>>1;
14        while(a>=0 && b<n && s[a]==s[b]) a--,b++,u[i]+=2;
15        if(i+u[i]>x+u[x]) x=i;
16    }
17    return *max_element(u.begin(),u.end());
18 }

```

6.9 模式匹配 (KMP+Z)

```

1 // 字符串前缀匹配 (KMP) 和后缀匹配 (Z-Function) By 猛犸也钻地 @ 2012.02.02
2
3 #include <vector>
4 #include <algorithm>
5 using namespace std;
6
7 // 计算字符串 s[0..n-1] 的前缀函数 (KMP), 复杂度 O(n)
8 // 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]
9 vector<int> calcP(const char *s, int n) {
10    vector<int> P(n);
11    for(int x=0,y=1; y<n; y++) {
12        while(x && s[x]!=s[y]) x=P[x-1];
13        if(s[x]==s[y]) P[y]=++x;
14    }
15    return P;
16 }
17
18 // 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
19 // 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]
20 vector<int> calcZ(const char *s, int n) {
21    vector<int> Z(n);
22    for(int i=1,x=0,y=0; i<n; i++) {
23        if(i<=y) Z[i]=min(y-i,Z[i-x]);
24        while(i+Z[i]<n && s[i+Z[i]]==s[Z[i]]) Z[i]++;
25        if(y<=i+Z[i]) x=i,y=i+Z[i];
26    }
27    return Z;

```

28 | }

6.10 模式匹配 (kmp)

```

1 //模式匹配,kmp 算法, 复杂度  $O(m+n)$ 
2 //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
3 //可更改元素类型, 更换匹配函数
4 const int MAXN = 10000;
5 #define _match(a, b) ((a) == (b))
6
7 template <class elemType>
8 int patMatch(int ls, const elemType *str, int lp, const elemType *pat) {
9     int fail[MAXN] = {-1}, i = 0, j;
10    for (j = 1; j < lp; j++) {
11        for (i = fail[j - 1]; i >= 0 && !_match(pat[i + 1], pat[j]); i = fail[i]);
12        fail[j] = (_match(pat[i + 1], pat[j]) ? i + 1 : -1);
13    }
14    for (i = j = 0; i < ls && j < lp; i++) {
15        if (_match(str[i], pat[j])) {
16            j++;
17        } else if (j) {
18            j = fail[j - 1] + 1;
19            i--;
20        }
21    }
22    return j == lp ? (i - lp) : -1;
23 }
24
25 // 统计次数
26
27 #define MAXN 10000
28 #define _match(a,b) ((a)==(b))
29 typedef char elem_t;
30
31 int pat_match(int ls,elem_t *str,int lp,elem_t *pat) {
32     int ret = 0;
33     int fail[MAXN]= {-1},i=0,j;
34     for (j=1; j<lp; j++) {
35         for (i=fail[j-1]; i>=0&& !_match(pat[i+1],pat[j]); i=fail[i]);
36         fail[j]=(_match(pat[i+1],pat[j])?i+1:-1);
37     }
38     for (i=j=0; i<ls; i++) {
39         if (_match(str[i],pat[j])) {
40             j++;
41             if (j == lp) {
42                 ++ret;
43                 j=fail[j-1]+1;
44             }
45         } else if (j)
46             j=fail[j-1]+1,i--;
47     }
48     return ret;
49 }
50

```

```

51
52 // 扩展 KMP, 复杂度  $O(m+n)$ 
53 // 传入匹配串 str 和模式串 pat 及长度, 返回 A[i] 值与 B[i] 值
54 // A[i] 表示 pat[i..m-1] 与 pat[0..m-1] 的最长公共前缀的长度
55 // B[i] 表示 str[i..n-1] 与 pat[0..m-1] 的最长公共前缀的长度
56
57 void extKMP(int n, const char str[], int m, const char pat[], int A[], int B[]) {
58     A[0] = m;
59     int ind = 0, k = 1;
60     while (ind + 1 < m && pat[ind + 1] == pat[ind]) ind++;
61     A[1] = ind;
62     for (int i = 2; i < m; i++) {
63         if (i <= k + A[k] - 1 && A[i - k] + i < k + A[k]) {
64             A[i] = A[i - k];
65         } else {
66             ind = max(0, k + A[k] - i);
67             while (ind + i < m && pat[ind + i] == pat[ind]) ind++;
68             A[i] = ind, k = i;
69         }
70     }
71     ind = 0, k = 0;
72     while (ind < n && str[ind] == pat[ind]) ind++;
73     B[0] = ind;
74     for (int i = 1; i < n; i++) {
75         if (i <= k + B[k] - 1 && A[i - k] + i < k + B[k]) {
76             B[i] = A[i - k];
77         } else {
78             ind = max(0, k + B[k] - i);
79             while (ind + i < n && ind < m && str[ind + i] == pat[ind]) ind++;
80             B[i] = ind, k = i;
81         }
82     }
83 }

```

Chapter 7

图论

7.1 NP 搜索

7.1.1 带权最大团

```

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

```

```

3    10.2.2000, Patric R. J. Ostergard,
4    patric.ostergard@hut.fi */
5
6    /* compile: gcc wclique.c -o wclique -O2 */
7
8    /* usage: wclique infile */
9
10   /* infile format: see http://www.tcs.hut.fi/~pat/wclique.html */
11
12   #include <stdio.h>
13   #include <sys/times.h>
14   #include <sys/types.h>
15
16   #define INT_SIZE (8*sizeof(int))
17   #define TRUE 1
18   #define FALSE 0
19   #define MAX_VERTEX 2000 /* maximum number of vertices */
20   #define MAX_WEIGHT 1000000 /* maximum weight of vertex */
21   #define is_edge(a,b) (bit[a][b/INT_SIZE]&(mask[b%INT_SIZE]))
22
23   int Vnbr,Enbr; /* number of vertices/edges */
24   int clique[MAX_VERTEX]; /* table for pruning */
25   int bit[MAX_VERTEX][MAX_VERTEX/INT_SIZE+1];
26   int wt[MAX_VERTEX];
27
28   int pos[MAX_VERTEX]; /* reordering function */
29   int set[MAX_VERTEX]; /* current clique */
30   int rec[MAX_VERTEX]; /* best clique so far */
31   int record; /* weight of best clique */
32   int rec_level; /* # of vertices in best clique */
33
34   unsigned mask[INT_SIZE];
35   void graph(); /* reads graph */
36
37   struct tms bf;
38   int timer1;
39   double timer11;
40
41   main (argc,argv)
42   int argc;
43   char *argv[];
44   {
45       int i,j,k,p;
46       int min_wt,max_nwt,wth;
47       int new[MAX_VERTEX],used[MAX_VERTEX];
48       int nwt[MAX_VERTEX];
49       int count;
50       FILE *infile;
51

```



```

52  /* read input */
53  if(argc < 2) {
54      printf("Usage: wclique infile\n");
55      exit(1);
56  }
57  if((infile=fopen(argv[1], "r"))==NULL)
58      fileerror();
59
60  /* initialize mask */
61  mask[0] = 1;
62  for(i=1; i<INT_SIZE; i++)
63      mask[i] = mask[i-1]<<1;
64
65  /* read graph */
66  graph(infile);
67
68  /* "start clock" */
69  times(&bf);
70  timer1 = bf.tms_utime;
71
72  /* order vertices */
73  for(i=0; i<Vnbr; i++) {
74      nwt[i] = 0;
75      for(j=0; j<Vnbr; j++)
76          if (is_edge(i,j)) nwt[i] += wt[j];
77  }
78  for(i=0; i<Vnbr; i++)
79      used[i] = FALSE;
80  count = 0;
81  do {
82      min_wt = MAX_WEIGHT+1;
83      max_nwt = -1;
84      for(i=Vnbr-1; i>=0; i--)
85          if ((!used[i])&&(wt[i]<min_wt))
86              min_wt = wt[i];
87      for(i=Vnbr-1; i>=0; i--) {
88          if(used[i]|| (wt[i]>min_wt)) continue;
89          if(nwt[i]>max_nwt) {
90              max_nwt = nwt[i];
91              p = i;
92          }
93      }
94      pos[count++] = p;
95      used[p] = TRUE;
96      for(j=0; j<Vnbr; j++)
97          if ((!used[j])&&(j!=p)&&(is_edge(p,j)))
98              nwt[j] -= wt[p];
99  } while(count<Vnbr);
100
101  /* main routine */
102  record = 0;
103  wth = 0;
104  for(i=0; i<Vnbr; i++) {
105      wth += wt[pos[i]];

```

```

106     sub(i,pos,0,0,wth);
107     clique[pos[i]] = record;
108     times(&bf);
109     timer11 = (bf.tms_ftime - timer1)/100.0;
110     printf("level = %3d(%d) best = %2d time = %8.2f\n",i+1,Vnbr,record,timer11);
111 }
112 printf("Record: ");
113 for(i=0; i<rec_level; i++)
114     printf ("%d ",rec[i]);
115 printf ("\n");
116 }
117
118 int sub(ct,table,level,weight,l_weight)
119 int ct,level,weight,l_weight;
120 int *table;
121 {
122     register int i,j,k;
123     int best;
124     int curr_weight,left_weight;
125     int newtable[MAX_VERTEX];
126     int *p1,*p2;
127
128     if(ct<=0) { /* 0 or 1 elements left; include these */
129         if(ct==0) {
130             set[level++] = table[0];
131             weight += l_weight;
132         }
133         if(weight>record) {
134             record = weight;
135             rec_level = level;
136             for (i=0; i<level; i++) rec[i] = set[i];
137         }
138         return 0;
139     }
140     for(i=ct; i>=0; i--) {
141         if((level==0)&&(i<ct)) return 0;
142         k = table[i];
143         if((level>0)&&(clique[k]<=(record-weight))) return 0; /* prune */
144         set[level] = k;
145         curr_weight = weight+wt[k];
146         l_weight -= wt[k];
147         if(l_weight<=(record-curr_weight)) return 0; /* prune */
148         p1 = newtable;
149         p2 = table;
150         left_weight = 0;
151         while (p2<table+i) {
152             j = *p2++;
153             if(is_edge(j,k)) {
154                 *p1++ = j;
155                 left_weight += wt[j];
156             }
157         }
158         if(left_weight<=(record-curr_weight)) continue;
159         sub(p1-newtable-1,newtable,level+1,curr_weight,left_weight);
160     }
161     return 0;

```

```

162 }
163
164 void graph(fp)
165 FILE *fp;
166 {
167     register int i,j,k;
168     int weight,degree,entry;
169
170     if(!fscanf(fp,"%d %d\n",&Vnbr,&Enbr))
171         fileerror();
172     for(i=0; i<Vnbr; i++) /* empty graph table */
173         for(j=0; j<Vnbr/INT_SIZE+1; j++)
174             bit[i][j] = 0;
175     for(i=0; i<Vnbr; i++) {
176         if(!fscanf(fp,"%d %d",&weight,&degree))
177             fileerror();
178         wt[i] = weight;
179         for(j=0; j<degree; j++) {
180             if(!fscanf(fp,"%d",&entry))
181                 fileerror();
182             bit[i][entry/INT_SIZE] |= mask[entry%INT_SIZE]; /* record edge */
183         }
184     }
185     fclose(fp);
186 }
187
188 int fileerror() {
189     printf("Error in graph file\n");
190     exit();
191 }

```

7.1.2 最大团 (n 小于 64)(faster)

```

1  /**
2   * WishingBone's ACM/ICPC Routine Library
3   *
4   * maximum clique solver
5   */
6
7  // 不知道怎么用……
8
9  #include <vector>
10
11 using std::vector;
12
13 // clique solver calculates both size and consitution of maximum clique
14 // uses bit operation to accelerate searching
15 // graph size limit is 63, the graph should be undirected
16 // can optimize to calculate on each component, and sort on vertex degrees
17 // can be used to solve maximum independent set

```

```

18 class clique {
19 public:
20     static const long long ONE = 1;
21     static const long long MASK = (1 << 21) - 1;
22     char *bits;
23     int n, size, cmax[63];
24     long long mask[63], cons;
25     // initiate Lookup table
26     clique() {
27         bits = new char[1 << 21];
28         bits[0] = 0;
29         for (int i = 1; i < 1 << 21; ++i) bits[i] = bits[i >> 1] + (i & 1);
30     }
31     ~clique() {
32         delete bits;
33     }
34     // search routine
35     bool search(int step, int size, long long more, long long con);
36     // solve maximum clique and return size
37     int sizeClique(vector<vector<int> > &mat);
38     // solve maximum clique and return constitution
39     vector<int> consClique(vector<vector<int> > &mat);
40 };
41
42 // search routine
43 // step is node id, size is current solution, more is available mask, cons is
44 // constitution mask
45 bool clique::search(int step, int size, long long more, long long cons) {
46     if (step >= n) {
47         // a new solution reached
48         this->size = size;
49         this->cons = cons;
50         return true;
51     }
52     long long now = ONE << step;
53     if ((now & more) > 0) {
54         long long next = more & mask[step];
55         if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >>
56             42] >= this->size
57             && size + cmax[step] > this->size) {
58             // the current node is in the clique
59             if (search(step + 1, size + 1, next, cons | now)) return true;
60         }
61     }
62     long long next = more & ~now;
63     if (size + bits[next & MASK] + bits[(next >> 21) & MASK] + bits[next >> 42]
64         > this->size) {
65         // the current node is not in the clique
66         if (search(step + 1, size, next, cons)) return true;
67     }
68     return false;

```

```

69 }
70
71 // solve maximum clique and return size
72 int clique::sizeClique(vector<vector<int> > &mat) {
73     n = mat.size();
74     // generate mask vectors
75     for (int i = 0; i < n; ++i) {
76         mask[i] = 0;
77         for (int j = 0; j < n; ++j) if (mat[i][j] > 0) mask[i] |= ONE << j;
78     }
79     size = 0;
80     for (int i = n - 1; i >= 0; --i) {
81         search(i + 1, 1, mask[i], ONE << i);
82         cmax[i] = size;
83     }
84     return size;
85 }
86
87 // solve maximum clique and return constitution
88 // calls sizeClique and restore cons
89 vector<int> clique::consClique(vector<vector<int> > &mat) {
90     sizeClique(mat);
91     vector<int> ret;
92     for (int i = 0; i < n; ++i) if ((cons & (ONE << i)) > 0) ret.push_back(i);
93     return ret;
94 }

```

7.1.3 最大团

```

1  const int maxn = 50;
2
3  void clique(int n, int mat[][maxn], int num, int U[], int size, int C[], int &_max, int \
4  ok) {
5      int i, j, k, tmp[maxn];
6      if (num == 0) {
7          if (size > _max) {
8              ok = 1;
9              _max = size;
10         }
11         return;
12     }
13     for (i = 0; i < num && !ok; ++i) {
14         if (size + num - i <= _max) return;
15         if (size + C[U[i]] <= _max) return;
16         for (k = 0, j = i + 1; j < num; ++j) if (mat[U[i]][U[j]])
17             tmp[k++] = U[j];
18         clique(n, mat, k, tmp, size + 1, C, _max, ok);
19     }
20 }
21
22 int max_clique(int n, int mat[][maxn]) {
23     int i, j, k, U[maxn], C[maxn], _max;
24     for (_max = 0, i = n - 1; i >= 0; --i) {
25         for (k = 0, j = i + 1; j < n; ++j) if (mat[i][j])
26             U[k++] = j;

```

```

27     clique(n, mat, k, U, 1, C, _max, 0);
28     C[i] = _max;
29 }
30 return _max;
31 }

```

7.2 匹配

7.2.1 一般图匹配 (Blossom)

```

1 // 一般图最大基数匹配 (Gabow) By 猛犸也钻地 @ 2012.05.02
2
3 #include <vector>
4 #include <cstring>
5 using namespace std;
6
7 class Blossom {
8 public:
9     static const int SIZE = 1005; // 最大结点个数
10    int cnt, mate[SIZE]; // mate[] 为配偶结点的编号, 没有匹配上的点为 -1
11    // 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
12    int gao(int n, const vector<int> e[]) { // 复杂度  $O(n^3)$ 
13        memset(mate, -1, sizeof(mate));
14        for(int z=0; z<n; z++) if(mate[z]<0) {
15            for(int i=0; i<n; i++) tag[i]=-2;
16            int q[SIZE], push=1, pop=0;
17            tag[q[0]=z]=-1;
18            while(push!=pop) {
19                int x=q[pop++%SIZE];
20                for(size_t i=0; i<e[x].size(); i++) {
21                    int y=e[x][i];
22                    if(mate[y]<0 && z!=y) {
23                        modify(mate[y]=x, y, n);
24                        i=push=pop=1234567890;
25                    } else if(tag[y]>=-1) {
26                        memset(at, 0, sizeof(at));
27                        travel(x, n), travel(y, n);
28                        for(int c=0; c<n; c++) if(at[c] && tag[c]<-1) {
29                            tag[c]=x+y*n+n;
30                            q[push++%SIZE]=c;
31                        }
32                    } else if(tag[mate[y]]<-1) {
33                        tag[mate[y]]=x;
34                        q[push++%SIZE]=mate[y];
35                    }
36                }
37            }
38        }
39        for(int i=cnt=0; i<n; i++) if(mate[i]>i) cnt++;
40        return cnt;
41    }
42 private:
43     int at[SIZE], tag[SIZE];

```

```

44 void modify(int x, int y, int n) {
45     int z=mate[x];
46     mate[x]=y;
47     if(z<0 || mate[z]!=x) return;
48     if(tag[x]<n) {
49         mate[z]=tag[x];
50         modify(mate[z],z,n);
51     } else {
52         y=tag[x]/n-1;
53         z=tag[x]%n;
54         modify(y,z,n);
55         modify(z,y,n);
56     }
57 }
58 void travel(int x, int n) {
59     int tmp[SIZE];
60     memcpy(tmp,mate,sizeof(tmp));
61     modify(x,x,n);
62     for(int i=0; i<n; i++)
63         if(mate[i]!=tmp[i]) at[i]^=1,mate[i]=tmp[i];
64 }
65 };

```

7.2.2 二分图最佳匹配 (kuhn munkras 邻接阵形式)

```

1 //二分图最佳匹配,kuhn munkras 算法, 邻接阵形式, 复杂度  $O(m*m*n)$ 
2 //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
3 //match1,match2 返回一个最佳匹配, 未匹配顶点  $match$  值为 -1
4 //一定注意  $m \leq n$ , 否则循环无法终止
5 //最小权匹配可将权值取相反数
6 #include <cstring>
7 const int MAXN = 310;
8 const int INF = 1000000000;
9 #define _clr(x) memset(x, 0xff, sizeof(int) * n)
10
11 int kuhnMunkras(int m, int n, int mat[][MAXN], int *match1, int *match2) {
12     int s[MAXN + 1], t[MAXN], l1[MAXN], l2[MAXN], p, q, ret = 0, i, j, k;
13     for (i = 0; i < m; i++) {
14         l1[i] = - INF;
15         for (j = 0; j < n; j++) {
16             l1[i] = mat[i][j] > l1[i] ? mat[i][j] : l1[i];
17         }
18     }
19     for (i = 0; i < n; l2[i++] = 0);
20     _clr(match1);
21     _clr(match2);
22     for (i = 0; i < m; i++) {
23         _clr(t);
24         for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
25             k = s[p];
26             for (j = 0; j < n && match1[i] < 0; j++) {
27                 if (l1[k] + l2[j] == mat[k][j] && t[j] < 0) {
28                     s[++q] = match2[j];
29                     t[j] = k;

```

```

30         if (s[q] < 0) {
31             for (p = j; p >= 0; j = p) {
32                 match2[j] = k = t[j];
33                 p = match1[k];
34                 match1[k] = j;
35             }
36         }
37     }
38 }
39 }
40 if (match1[i] < 0) {
41     i--;
42     p = INF;
43     for (k = 0; k <= q; k++) {
44         for (j = 0; j < n; j++) {
45             if (t[j] < 0 && l1[s[k]] + l2[j] - mat[s[k]][j] < p) {
46                 p = l1[s[k]] + l2[j] - mat[s[k]][j];
47             }
48         }
49     }
50     for (j = 0; j < n; j++) {
51         l2[j] += t[j] < 0 ? 0 : p;
52     }
53     for (k = 0; k <= q; k++) {
54         l1[s[k]] -= p;
55     }
56 }
57 }
58 for (i = 0; i < m; i++) {
59     ret += mat[i][match1[i]];
60 }
61 return ret;
62 }

```

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

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 2005;
5 const int INF = 1e9 + 7;
6
7 struct KuhnMunkres {
8     int sum, mx[N], my[N], lx[N], ly[N], sx[N], sy[N];
9     bool vx[N], vy[N];
10    int gao(int n, int e[N][N]) {
11        fill_n(mx, n, -1);
12        fill_n(my, n, -1);
13        fill_n(ly, n, 0);
14        for (int i = 0; i < n; ++i) lx[i] = *max_element(e[i], e[i]+n);
15        for (int x = 0, y, z, w; x < n; ++x) {
16            fill_n(vx, n, false);
17            fill_n(vy, n, false);
18            queue<int> q;
19            vx[x] = true;
20            for (int i = 0; i < n; ++i) {
21                sx[i] = x, sy[i] = lx[x] + ly[i] - e[x][i];

```



```

22         if (!sy[i]) q.push(i), vy[i] = true;
23     }
24     while (1) {
25         if (q.empty()) {
26             int delta = *min_element(sy, sy+n);
27             for (int i = 0; i < n; ++i) {
28                 if (vx[i]) lx[i] -= delta;
29                 if (vy[i]) ly[i] += delta;
30                 else if (!sy[i] - delta) q.push(i), vy[i] = true;
31             }
32         } else {
33             y = q.front();
34             q.pop();
35             if (!~my[y]) break;
36             vx[z = my[y]] = vy[y] = true, sy[y] = INF;
37             for (int i = 0; i < n; ++i) if (!vy[i]) {
38                 int d = lx[z] + ly[i] - e[z][i];
39                 if (sy[i] > d) sy[i] = d, sx[i] = z;
40                 if (!sy[i]) q.push(i), vy[i] = true;
41             }
42         }
43     }
44     for (; ~y; y = w) z = sx[y], w = mx[z], mx[z] = y, my[y] = z;
45 }
46 for (int i = sum = 0; i < n; ++i) sum += lx[i] + ly[i];
47 return sum;
48 }
49 } km;
50
51 int n, e[N][N];
52
53 int main() {
54     scanf("%d", &n);
55     for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j) e[i][j] = (i+1)*(j+1);
56     int ans = km.gao(n, e);
57     for (int i = 0; i < n; ++i) {
58         assert(km.mx[i] == i && km.my[i] == i);
59         ans -= (i+1)*(i+1);
60     }
61     assert(!ans);
62 }

```

7.2.4 二分图最大匹配 (hopcroft kart 邻接表形式)

```

1 // Hopcroft_Karp matching algorithm, 图的大小为 n 和 m, 返回最大匹配数
2 // vector 存储, 复杂度 O(sqrt(V)*E)
3 // match1 和 match2 为最大匹配, 未匹配节点 match1 为 -1, match2 为 n
4 // 每次使用之前将边信息存入 E
5
6 #include <cstring>
7 #include <vector>
8 using namespace std;
9
10 const int MAXN = 50005;
11 const int MAXM = 200005;

```

```

12
13 int n, m;
14 int match1[MAXN], match2[MAXN];
15 int Q[MAXN], D1[MAXN], D2[MAXN];
16 vector <int> E[MAXN];
17
18 inline bool bfs() {
19     int s = 0, t = 0, u, v;
20     memset(D1, -1, sizeof(D1));
21     memset(D2, -1, sizeof(D2));
22     for (int i = 0; i < n; i++)
23         if (match1[i] == -1)
24             Q[t++] = i, D1[i] = 0;
25     while (s != t)
26         if ((u = Q[s++]) != n)
27             for (int i = 0; i < (int)E[u].size(); i++)
28                 if (D2[v = E[u][i]] == -1) {
29                     D2[v] = D1[u] + 1;
30                     if (D1[match2[v]] == -1)
31                         D1[Q[t++] = match2[v]] = D2[v] + 1;
32                 }
33     return D1[n] != -1;
34 }
35
36 bool dfs(int u) {
37     for (int i = 0, v; i < (int)E[u].size(); i++)
38         if (D2[v = E[u][i]] == D1[u] + 1 && (D2[v] == -1 || dfs(match\
39 2[v]))) {
40             match1[u] = v;
41             match2[v] = u;
42             D1[u] = -1;
43             return true;
44         }
45     D1[u] = -1;
46     return false;
47 }
48
49 inline int hopcroft_karp() {
50     memset(match1, -1, sizeof(match1));
51     for (int i = 0; i < m; i++)
52         match2[i] = n;
53     int ret = 0;
54     for (int i = 0; i < n; i++)
55         for (int j = 0, u; j < E[i].size(); j++)
56             if (match2[u = E[i][j]] == n) {
57                 match1[i] = u;
58                 match2[u] = i;
59                 ret++;
60                 break;
61             }
62     while (bfs())
63         for (int i = 0; i < n; i++)
64             if (match1[i] == -1 && dfs(i))
65                 ret++;
66     return ret;
67 }

```

7.2.5 二分图最大匹配 (hungary bfs 邻接阵形式)

```

1 //二分图最大匹配,hungary 算法, 邻接阵形式, 复杂度  $O(n*n*m)$ 
2 //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
3 //match1,match2 返回一个最大匹配, 未匹配顶点  $match$  值为 -1
4 #include <cstring>
5 const int MAXN = 310;
6 #define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)
7
8 int hungary(int n, int m, const bool mat[][MAXN], int *match1, int *match2) {
9     int s[MAXN + 1], t[MAXN], p, q, ret = 0, i, j, k;
10    _clr(match1);
11    _clr(match2);
12    for (i = 0; i < n; ret += (match1[i++] >= 0)) {
13        _clr(t);
14        for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
15            k = s[p];
16            for (j = 0; j < m && match1[i] < 0; j++) {
17                if (mat[k][j] && t[j] < 0) {
18                    s[++q] = match2[j];
19                    t[j] = k;
20                    if (s[q] < 0) {
21                        for (p = j; p >= 0; j = p) {
22                            match2[j] = k = t[j];
23                            p = match1[k];
24                            match1[k] = j;
25                        }
26                    }
27                }
28            }
29        }
30    }
31    return ret;
32 }

```

7.2.6 二分图最大匹配 (hungary dfs 邻接阵形式)

```

1 //二分图最大匹配,hungary dfs 算法, 邻接阵形式, 复杂度  $O(n*n*m)$ 
2 //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
3 //match1,match2 返回一个最大匹配, 未匹配顶点  $match$  值为 -1
4 #define _clr(x) memset(x, 0xff, sizeof(int) * MAXN)
5 const int MAXN = 200;
6 int mk[MAXN], match1[MAXN], match2[MAXN], mat[MAXN][MAXN], n, m;
7 int path(int i) {
8     for (int j = 0; j < m; j++)
9         if (mat[i][j] && !(mk[j]++) && (match2[j] < 0 || path(match2[j]))) {
10             match1[i] = j;
11             match2[j] = i;
12             return 1;
13         }
14     return 0;
15 }
16 int hungary() {
17     int res(0);

```

```

18     _clr(match1);
19     _clr(match2);
20     for (int i = 0; i < n; i++)
21         if (match1[i] < 0) {
22             memset(mk, 0, sizeof(mk));
23             res += path(i);
24         }
25     return res;
26 }

```

7.3 应用

7.3.1 2-sat

```

1 // O(N + M)
2 // 调用 init() 初始化, 传入节点个数, 节点标号必须从 0 开始
3 // 调用 add_edge() 添加边, 调用 add_var() 添加变量的初始值
4 // 调用 solve() 求解 2sat 如果有解存在 mark 中
5 // 对于变量 x, x*2 表示 x, x * 2 + 1 表示 ~x, mark[x << 1] 表示 x 的值
6 struct TwoSAT {
7     static const int MAXN = 200000 + 10;
8     vector<int> G[MAXN], SCC[MAXN];
9     int low[MAXN], dfn[MAXN], stk[MAXN];
10    int col[MAXN], mark[MAXN];
11    int scc_cnt, top, sz, n;
12    void init(int n) {
13        this->n = n;
14        scc_cnt = 0;
15        for (int i = 0; i < n * 2; ++ i) {
16            G[i].clear();
17            SCC[i].clear();
18        }
19    }
20    void dfs(int x) {
21        low[x] = dfn[x] = ++ sz;
22        stk[top ++] = x;
23        mark[x] = true;
24        for (int i = 0, y; i < (int)G[x].size(); ++ i) {
25            if (!dfn[y = G[x][i]]) {
26                dfs(y);
27                low[x] = min(low[x], low[y]);
28            } else if (mark[y]) low[x] = min(low[x], dfn[y]);
29        }
30        if (dfn[x] == low[x]) {
31            SCC[scc_cnt ++].clear();
32            for (int y; ; ) {
33                mark[y = stk[-- top]] = false;
34                SCC[scc_cnt - 1].push_back(y);
35                col[y] = scc_cnt - 1;
36                if (y == x) break;
37            }
38        }
39    }
40    inline int get_val(int x) {

```

```

41     int r = (x & 1) ? x ^ 1 : x;
42     if (mark[r] == -1) return -1;
43     return (x & 1) ? !mark[r] : mark[r];
44 }
45 void construct() {
46     for (int i = 0; i < n * 2; ++ i) mark[i] = -1;
47     for (int i = 0, j, val; i < scc_cnt; ++ i) {
48         for (val = 1, j = 0; j < (int)SCC[i].size(); ++ j) {
49             int cur = SCC[i][j];
50             if (get_val(cur) == 0) val = 0;
51             for (int k = 0; k < (int)G[cur].size(); ++ k)
52                 if (get_val(G[cur][k]) == 0) val = 0;
53             if (val == 0) break;
54         }
55         for (j = 0; j < (int)SCC[i].size(); ++ j) {
56             if (SCC[i][j] & 1) mark[SCC[i][j] ^ 1] = !val;
57             else mark[SCC[i][j]] = val;
58         }
59     }
60 }
61 bool solve() {
62     for (int i = 0; i < 2 * n; ++ i) mark[i] = false, dfn[i] = 0;
63     top = sz = 0;
64     for (int i = 0; i < 2 * n; ++ i)
65         if (!dfn[i]) dfs(i);
66     for (int i = 0; i < 2 * n; i += 2)
67         if (col[i] == col[i ^ 1]) return false;
68     construct();
69     return true;
70 }
71 void add_edge(int x, int y) { // x -> y, 0 <= x, y < 2 * n
72     G[x].push_back(y);
73 }
74 void add_var(int x, int xv) { // x = xv, 0 <= x < n
75     x = x << 1 | xv;
76     G[x ^ 1].push_back(x);
77 }
78 };

```

7.3.2 前序表转化

```

1 //将用边表示的树转化为前序表示的树
2 //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
3 //pre[] 返回前序表, map[] 返回前序表中的节点到原来节点的映射
4 const int MAXN = 10000;
5 struct Node {
6     int to;
7     Node *next;
8 };
9
10 void prenode(int n, Node *list[], int *pre, int *map, int *v, int now, int last, int &id\
11 ) {
12     Node *t;
13     int p = id++;

```

```

14     for (v[map[p] = now] = 1, pre[p] = last; list[now];) {
15         t = list[now];
16         list[now] = t->next;
17         if (!v[t->to]) {
18             prenode(n, list, pre, map, v, t->to, p, id);
19         }
20     }
21 }
22
23 void makepre(int n, Node *list[], int *pre, int *map) {
24     int v[MAXN], id = 0, i;
25     for (i = 0; i < n; v[i++] = 0);
26     prenode(n, list, pre, map, v, 0, -1, id);
27 }

```

7.3.3 拓扑排序 (邻接阵形式)

```

1 //拓扑排序, 邻接阵形式, 复杂度  $O(n^2)$ 
2 //如果无法完成排序, 返回 0, 否则返回 1, ret 返回有序点列
3 //传入图的大小 n 和邻接阵 mat, 不相邻点边权 0
4 const int MAXN = 100;
5
6 bool toposort(int n, int mat[][MAXN], int *ret) {
7     int d[MAXN], i, j, k;
8     for (i = 0; i < n; i++) {
9         for (d[i] = j = 0; j < n; d[i] += mat[j][i]);
10    }
11    for (k = 0; k < n; ret[k++] = i) {
12        for (i = 0; d[i] && i < n; i++);
13        if (i == n) {
14            return false;
15        }
16        for (d[i] = -1, j = 0; j < n; j++) {
17            d[j] -= mat[i][j];
18        }
19    }
20    return true;
21 }

```

7.3.4 无向图全局最小割

```

1 // 无向图全局最小割 (Stoer-Wagner) By 猛犸也钻地 @ 2012.08.22
2
3 #include <vector>
4 #include <algorithm>
5 using namespace std;
6
7 class StoerWagner {
8 public:
9     typedef int VAL; // 权值的类型
10    static const int SIZE = 505; // 最大结点数
11    static const VAL INF = 1000000007; // 最大权值之和
12    VAL sum, e[SIZE][SIZE];

```

```

13 // 传入结点个数 n 及权值矩阵 a[SIZE][SIZE], 返回无向图全局最小割的边权之和 sum
14 // 对于矩阵 a[SIZE][SIZE] 中不存在的边, 权值设为 0
15 int gao(int n, const VAL a[SIZE][SIZE]) {
16     vector<int> v, idx(n);
17     for(int i=0; i<n; i++) copy(a[i], a[i]+n, e[idx[i]=i]);
18     for(int i=0; i<n; i++) e[i][i]=0;
19     for(sum=INF; idx.size()>=2; n=idx.size()) {
20         vector<VAL> s(n);
21         for(int i=0; i<n; i++) v.push_back(i);
22         int p=0, t=0;
23         while(v.size()) {
24             int m=v.size(), x=-1;
25             for(int i=0; i<m; i++) if(x<0 || s[x]<s[i]) x=i;
26             for(int i=0; i<m; i++) s[i]+=e[idx[v[x]]][idx[v[i]]];
27             v.erase(v.begin()+x);
28             s.erase(s.begin()+x);
29             swap(p=v[x], t);
30         }
31         VAL now=0;
32         for(int i=0; i<n; i++) if(i!=t) now+=e[idx[t]][idx[i]];
33         for(int i=0; i<n; i++) {
34             e[idx[i]][idx[p]]+=e[idx[i]][idx[t]];
35             e[idx[p]][idx[i]]+=e[idx[t]][idx[i]];
36         }
37         idx.erase(idx.begin()+t);
38         sum=min(sum, now);
39     }
40     return sum;
41 }
42 };

```

7.3.5 无向图最小环

```

1 // 无向图最小环 (Floyd) By 猛犸也钻地 @ 2012.09.13
2
3 #include <vector>
4 #include <cstring>
5 using namespace std;
6
7 class Floyd {
8 public:
9     typedef int VAL; // 权值的类型
10    static const int SIZE = 105;
11    vector<int> path;
12    VAL len[SIZE][SIZE], ans;
13    int src[SIZE][SIZE];
14    // 传入结点个数 n 及权值矩阵 a[SIZE][SIZE], 返回最小环的长度 ans, 方案记在 path 中
15    // 对于矩阵 a[SIZE][SIZE] 中不存在的边, 权值设为 1e9+7 或 0x7F7F7F7F 之类的极大值
16    VAL gao(int n, const VAL a[SIZE][SIZE]) {
17        ans=1e9+7; // 若最后的返回值大于等于 1e9, 则不存在最小环
18        memset(src, -1, sizeof(src));
19        memcpy(len, a, sizeof(len));
20        for(int k=0; k<n; k++) {

```

```

21     for(int i=0; i<k; i++) for(int j=i+1; j<k; j++) {
22         VAL tmp=a[k][i]+a[j][k];
23         if(len[i][j]>=ans-tmp) continue;
24         path.clear();
25         getpath(i,j);
26         path.push_back(k);
27         path.push_back(i);
28         ans=tmp+len[i][j];
29     }
30     for(int i=0; i<n; i++) for(int j=0; j<n; j++) {
31         VAL tmp=len[i][k]+len[k][j];
32         if(tmp>=len[i][j]) continue;
33         len[i][j]=tmp;
34         src[i][j]=k;
35     }
36 }
37 return ans;
38 }
39 private:
40 void getpath(int i, int j) {
41     int k=src[i][j];
42     if(~k) {
43         getpath(i,k);
44         getpath(k,j);
45     } else {
46         path.push_back(j);
47     }
48 }
49 };

```

7.3.6 最佳边割集

```

1 //最佳边割集
2 const int MAXN = 100;
3 const int INF = 1000000000;
4
5 int maxFlow(int n, int mat[][MAXN], int source, int sink) {
6     int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
7     while (true) {
8         for (i = 0; i < n; i++) {
9             v[i] = c[i] = 0;
10        }
11        for (c[source] = INF;;) {
12            for (j = -1, i = 0; i < n; i++) {
13                if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
14                    j = i;
15                }
16            }
17            if (j < 0) {
18                return ret;
19            }
20            if (j == sink) {
21                break;
22            }
23            for (v[j] = 1, i = 0; i < n; i++) {
24                if (mat[j][i] > c[i] && c[j] > c[i]) {
25                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];

```



```

26         p[i] = j;
27     }
28 }
29 }
30 for (ret += j = c[i = sink]; i != source; i = p[i]) {
31     mat[p[i]][i] -= j;
32     mat[i][p[i]] += j;
33 }
34 }
35 }
36
37 int bestEdgeCut(int n, int mat[][MAXN], int source, int sink, int set[][2], int &mincost\
38 ) {
39     int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;
40     if (source == sink) {
41         return -1;
42     }
43     for (i = 0; i < n; i++) {
44         for (j = 0; j < n; j++) {
45             m0[i][j] = mat[i][j];
46         }
47     }
48     for (i = 0; i < n; i++) {
49         for (j = 0; j < n; j++) {
50             m[i][j] = m0[i][j];
51         }
52     }
53     mincost = last = maxFlow(n, m, source, sink);
54     for (k = 0; k < n && last; k++) {
55         for (l = 0; l < n && last; l++) {
56             if (m0[k][l]) {
57                 for (i = 0; i < n + n; i++) {
58                     for (j = 0; j < n + n; j++) {
59                         m[i][j] = m0[i][j];
60                     }
61                 }
62                 m[k][l] = 0;
63                 if (maxFlow(n, m, source, sink) == last - mat[k][l]) {
64                     set[ret][0] = k;
65                     set[ret++][1] = l;
66                     m0[k][l] = 0;
67                     last -= mat[k][l];
68                 }
69             }
70         }
71     }
72     return ret;
73 }

```

7.3.7 最佳顶点割集

```

1 //最佳顶点割集
2 const int MAXN = 100;
3 const int INF = 1000000000;
4
5 int maxFlow(int n, int mat[][MAXN], int source, int sink) {
6     int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;

```

```

7   while (true) {
8       for (i = 0; i < n; i++) {
9           v[i] = c[i] = 0;
10      }
11      for (c[source] = INF;;) {
12          for (j = -1, i = 0; i < n; i++) {
13              if (!v[i] && c[i] && (j == -1 || c[i] > c[j])) {
14                  j = i;
15              }
16          }
17          if (j < 0) {
18              return ret;
19          }
20          if (j == sink) {
21              break;
22          }
23          for (v[j] = 1, i = 0; i < n; i++) {
24              if (mat[j][i] > c[i] && c[j] > c[i]) {
25                  c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
26                  p[i] = j;
27              }
28          }
29      }
30      for (ret += j = c[i = sink]; i != source; i = p[i]) {
31          mat[p[i]][i] -= j;
32          mat[i][p[i]] += j;
33      }
34  }
35 }
36
37 int bestVertexCut(int n, int mat[][MAXN], int *cost, int source, int sink, int *set, int\
38 &mincost) {
39     int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;
40     if (source == sink || mat[source][sink]) {
41         return -1;
42     }
43     for (i = 0; i < n + n; i++) {
44         for (j = 0; j < n + n; j++) {
45             m0[i][j] = 0;
46         }
47     }
48     for (i = 0; i < n; i++) {
49         for (j = 0; j < n; j++) {
50             if (mat[i][j]) {
51                 m0[i][n + j] = INF;
52             }
53         }
54     }
55     for (i = 0; i < n; i++) {
56         m0[n + i][i] = cost[i];
57     }
58     for (i = 0; i < n + n; i++) {
59         for (j = 0; j < n + n; j++) {
60             m[i][j] = m0[i][j];
61         }
62     }
63     mincost = last = maxFlow(n + n, m, source, n + sink);
64     for (k = 0; k < n && last; k++) {

```

```

65     if (k != source && k != sink) {
66         for (i = 0; i < n + n; i++) {
67             for (j = 0; j < n + n; j++) {
68                 m[i][j] = m0[i][j];
69             }
70         }
71         m[n + k][k] = 0;
72         if (maxFlow(n + n, m, source, n + sink) == last - cost[k]) {
73             set[ret++] = k;
74             m0[n + k][k] = 0;
75             last -= cost[k];
76         }
77     }
78 }
79 return ret;
80 }

```

7.3.8 最小路径覆盖

```

1 //最小路径覆盖,  $O(n^3)$ 
2 //求解最小的路径覆盖图中所有点, 有向图无向图均适用
3 //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
4 //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)
5 #include <cstring>
6 const int MAXN = 310;
7 #define _clr(x) memset(x, 0xff, sizeof(int) * n)
8
9 int hungary(int n, const bool mat[][MAXN], int *match1, int *match2) {
10     int s[MAXN], t[MAXN], p, q, ret = 0, i, j, k;
11     _clr(match1);
12     _clr(match2);
13     for (i = 0; i < n; ret += (match1[i++] >= 0)) {
14         _clr(t);
15         for (s[p = q = 0] = i; p <= q && match1[i] < 0; p++) {
16             for (k = s[p], j = 0; j < n && match1[i] < 0; j++) {
17                 if (mat[k][j] && t[j] < 0) {
18                     s[++q] = match2[j];
19                     t[j] = k;
20                     if (s[q] < 0) {
21                         for (p = j; p >= 0; j = p) {
22                             match2[j] = k = t[j];
23                             p = match1[k];
24                             match1[k] = j;
25                         }
26                     }
27                 }
28             }
29         }
30     }
31     return ret;
32 }
33
34 inline int pathCover(int n, const bool mat[][MAXN], int *pre, int *next) {
35     return n - hungary(n, mat, next, pre);
36 }

```

7.3.9 最小边割集

```

1 //最小边割集
2 const int MAXN = 100;
3 const int INF = 1000000000;
4
5 int maxFlow(int n, int mat[][MAXN], int source, int sink) {
6     int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
7     while (true) {
8         for (i = 0; i < n; i++) {
9             v[i] = c[i] = 0;
10        }
11        for (c[source] = INF;;) {
12            for (j = -1, i = 0; i < n; i++) {
13                if (!v[i] && c[i] && (j == -1 || c[i] < c[j])) {
14                    j = i;
15                }
16            }
17            if (j < 0) {
18                return ret;
19            }
20            if (j == sink) {
21                break;
22            }
23            for (v[j] = 1, i = 0; i < n; i++) {
24                if (mat[j][i] > c[i] && c[j] > c[i]) {
25                    c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
26                    p[i] = j;
27                }
28            }
29        }
30        for (ret += j = c[i = sink]; i != source; i = p[i]) {
31            mat[p[i]][i] -= j;
32            mat[i][p[i]] += j;
33        }
34    }
35 }
36
37 int minEdgeCut(int n, int mat[][MAXN], int source, int sink, int set[][2]) {
38     int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, l, ret = 0, last;
39     if (source == sink) {
40         return -1;
41     }
42     for (i = 0; i < n; i++) {
43         for (j = 0; j < n; j++) {
44             m0[i][j] = (mat[i][j] != 0);
45         }
46     }
47     for (i = 0; i < n; i++) {
48         for (j = 0; j < n; j++) {
49             m[i][j] = m0[i][j];
50         }
51     }
52     last = maxFlow(n, m, source, sink);
53     for (k = 0; k < n && last; k++) {
54         for (l = 0; l < n && last; l++) {
55             if (m0[k][l]) {
56                 for (i = 0; i < n + n; i++) {

```

```

57         for (j = 0; j < n + n; j++) {
58             m[i][j] = m0[i][j];
59         }
60     }
61     m[k][1] = 0;
62     if (maxFlow(n, m, source, sink) < last) {
63         set[ret][0] = k;
64         set[ret++][1] = 1;
65         m0[k][1] = 0;
66         last--;
67     }
68 }
69 }
70 }
71 return ret;
72 }

```

7.3.10 最小顶点割集

```

1  //最小顶点割集
2  const int MAXN = 100;
3  const int INF = 1000000000;
4
5  int maxFlow(int n, int mat[][MAXN], int source, int sink) {
6      int v[MAXN], c[MAXN], p[MAXN], ret = 0, i, j;
7      while (true) {
8          for (i = 0; i < n; i++) {
9              v[i] = c[i] = 0;
10         }
11         for (c[source] = INF;;) {
12             for (j = -1, i = 0; i < n; i++) {
13                 if (!v[i] && c[i] && (j == -1 || c[i] < c[j])) {
14                     j = i;
15                 }
16             }
17             if (j < 0) {
18                 return ret;
19             }
20             if (j == sink) {
21                 break;
22             }
23             for (v[j] = 1, i = 0; i < n; i++) {
24                 if (mat[j][i] > c[i] && c[j] > c[i]) {
25                     c[i] = mat[j][i] < c[j] ? mat[j][i] : c[j];
26                     p[i] = j;
27                 }
28             }
29         }
30         for (ret += j = c[i = sink]; i != source; i = p[i]) {
31             mat[p[i]][i] -= j;
32             mat[i][p[i]] += j;
33         }
34     }
35 }
36
37 int minVertexCut(int n, int mat[][MAXN], int source, int sink, int *set) {
38     int m0[MAXN][MAXN], m[MAXN][MAXN], i, j, k, ret = 0, last;

```

```

39     if (source == sink || mat[source][sink]) {
40         return - 1;
41     }
42     for (i = 0; i < n + n; i++) {
43         for (j = 0; j < n + n; j++) {
44             m0[i][j] = 0;
45         }
46     }
47     for (i = 0; i < n; i++) {
48         for (j = 0; j < n; j++) {
49             if (mat[i][j]) {
50                 m0[i][n + j] = INF;
51             }
52         }
53     }
54     for (i = 0; i < n; i++) {
55         m0[n + i][i] = 1;
56     }
57     for (i = 0; i < n + n; i++) {
58         for (j = 0; j < n + n; j++) {
59             m[i][j] = m0[i][j];
60         }
61     }
62     last = maxFlow(n + n, m, source, n + sink);
63     for (k = 0; k < n && last; k++) {
64         if (k != source && k != sink) {
65             for (i = 0; i < n + n; i++) {
66                 for (j = 0; j < n + n; j++) {
67                     m[i][j] = m0[i][j];
68                 }
69             }
70             m[n + k][k] = 0;
71             if (maxFlow(n + n, m, source, n + sink) < last) {
72                 set[ret++] = k;
73                 m0[n + k][k] = 0;
74                 last--;
75             }
76         }
77     }
78     return ret;
79 }

```

7.3.11 树的优化算法

```

1  const int MAXN = 1000;
2
3  //最大顶点独立集
4  int maxNodeIndependent(int n, int *pre, int *set) {
5      int c[MAXN], i, ret = 0;
6      for (i = 0; i < n; i++) {
7          c[i] = set[i] = 0;
8      }
9      for (i = n - 1; i >= 0; i--) {
10         if (!c[i]) {
11             set[i] = 1;
12             if (pre[i] != -1) {
13                 c[pre[i]] = 1;

```

```

14         }
15         ret++;
16     }
17 }
18 return ret;
19 }
20
21 //最大边独立集
22 int maxEdgeIndependent(int n, int *pre, int *set) {
23     int c[MAXN], i, ret = 0;
24     for (i = 0; i < n; i++) {
25         c[i] = set[i] = 0;
26     }
27     for (i = n - 1; i >= 0; i--) {
28         if (!c[i] && pre[i] != -1 && !c[pre[i]]) {
29             set[i] = 1;
30             c[pre[i]] = 1;
31             ret++;
32         }
33     }
34     return ret;
35 }
36
37 //最小顶点覆盖集
38 int minNodeCover(int n, int *pre, int *set) {
39     int c[MAXN], i, ret = 0;
40     for (i = 0; i < n; i++) {
41         c[i] = set[i] = 0;
42     }
43     for (i = n - 1; i >= 0; i--) {
44         if (!c[i] && pre[i] != -1 && !c[pre[i]]) {
45             set[i] = 1;
46             c[pre[i]] = 1;
47             ret++;
48         }
49     }
50     return ret;
51 }
52
53 //最小顶点支配集
54 int minNodeDominant(int n, int *pre, int *set) {
55     int c[MAXN], i, ret = 0;
56     for (i = 0; i < n; i++) {
57         c[i] = set[i] = 0;
58     }
59     for (i = n - 1; i >= 0; i--) {
60         if (!c[i] && (pre[i] == -1 || !set[pre[i]])) {
61             if (pre[i] != -1) {
62                 set[pre[i]] = 1;
63                 c[pre[i]] = 1;
64                 if (pre[pre[i]] != -1) {
65                     c[pre[pre[i]]] = 1;
66                 }
67             } else {
68                 set[i] = 1;
69             }
70         }
71     }
72     return ret;
73 }

```

```

70         ret++;
71     }
72 }
73 return ret;
74 }

```

7.3.12 欧拉回路 (邻接阵形式)

```

1 //求欧拉回路或欧拉路, 邻接阵形式, 复杂度  $O(n^2)$ 
2 //返回路径长度, path 返回路径 (有向图时得到的是反向路径)
3 //传入图的大小 n 和邻接阵 mat, 不相邻点边权 0
4 //可以有自环与重边, 分为无向图和有向图
5
6 const int MAXN = 100;
7
8 void findPathU(int n, int mat[][MAXN], int now, int &step, int *path) {
9     int i;
10    for (i = n - 1; i >= 0; i--) {
11        while (mat[now][i]) {
12            mat[now][i]--;
13            mat[i][now]--;
14            findPathU(n, mat, i, step, path);
15        }
16    }
17    path[step++] = now;
18 }
19
20 void findPathD(int n, int mat[][MAXN], int now, int &step, int *path) {
21     int i;
22     for (i = n - 1; i >= 0; i--) {
23         while (mat[now][i]) {
24             mat[now][i]--;
25             findPathD(n, mat, i, step, path);
26         }
27     }
28     path[step++] = now;
29 }
30
31 int euclidPath(int n, int mat[][MAXN], int start, int *path) {
32     int ret = 0;
33     findPathU(n, mat, start, ret, path);
34     // findPathD(n, mat, start, ret, path);
35     return ret;
36 }

```

7.4 生成树

7.4.1 多源最小树形图 (邻接阵形式)

```

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

```



```

3 //传入图的大小 n 和邻接阵 mat, 不相邻点边权 inf
4 //可更改边权的类型,pre[] 返回树的构造, 用父结点表示
5 //传入时 pre[] 数组清零, 用 -1 标出可能的源点
6 #include <string.h>
7 #define MAXN 120
8 #define inf 1000000000
9 typedef int elem_t;
10 elem_t edmonds(int n,elem_t mat[][MAXN*2],int *pre) {
11     elem_t ret=0;
12     int c[MAXN*2][MAXN*2],l[MAXN*2],p[MAXN*2],m=n,t,i,j,k;
13     for (i=0; i<n; l[i]=i,i++);
14     do {
15         memset(c,0,sizeof(c)),memset(p,0xff,sizeof(p));
16         for (t=m,i=0; i<m; c[i][i]=1,i++);
17         for (i=0; i<t; i++)
18             if (l[i]==i&&pre[i]!=-1) {
19                 for (j=0; j<m; j++)
20                     if
21                         (l[j]==j&&i!=j&&mat[j][i]<inf&&(p[i]==-1|mat[j][i]<mat[p[i]][i]))
22                         p[i]=j;
23                 if ((pre[i]=p[i])!=-1)
24                     return -1;
25                 if (c[i][p[i]]) {
26                     for (j=0; j<m; mat[j][m]=mat[m][j]=inf,j++);
27                     for (k=i; l[k]!=m; l[k]=m,k=p[k])
28                         for (j=0; j<m; j++)
29                             if (l[j]==j) {
30                                 if (mat[j][k]-mat[p[k]][k]<mat[j][m])
31                                     mat[j][m]=mat[j][k]-mat[p[k]][k];
32                                 if (mat[k][j]<mat[m][j])
33                                     mat[m][j]=mat[k][j];
34                             }
35                     c[m][m]=1,l[m]=m,m++;
36                 }
37                 for (j=0; j<m; j++)
38                     if (c[i][j])
39                         for (k=p[i]; k!=-1&&l[k]==k; c[k][j]=1,k=p[k]);
40             }
41     } while (t<m);
42     for (; m-->n; pre[k]=pre[m])
43         for (i=0; i<m; i++)
44             if (l[i]==m) {
45                 for (j=0; j<m; j++)
46                     if (pre[j]==m&&mat[i][j]==mat[m][j])
47                         pre[j]=i;
48                 if (mat[pre[m]][m]==mat[pre[m]][i]-mat[pre[i]][i])
49                     k=i;
50             }
51     for (i=0; i<n; i++)
52         if (pre[i]!=-1)
53             ret+=mat[pre[i]][i];
54     return ret;
55 }

```

7.4.2 最小生成树 (kruskal 邻接表形式)

```

1 //无向图最小生成树,kruskal 算法, 邻接表形式, 复杂度  $O(m\log m)$ 
2 //返回最小生成树的长度, 传入图的大小  $n$  和邻接表  $List$ 
3 //可更改边权的类型,  $edge[][2]$  返回树的构造, 用边集表示
4 //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
5 #include <cstring>
6 const int MAXN = 200;
7 const int INF = 1000000000;
8
9 #define _run(x) for(; p[t = x]; x = p[x], p[t] = (p[x] ? p[x] : x))
10 #define _run_both _run(i); _run(j)
11
12 class DSet {
13 public:
14     int p[MAXN], t;
15     void init() {
16         memset(p, 0, sizeof(p));
17     }
18     void setFriend(int i, int j) {
19         _run_both;
20         p[i] = (i == j ? 0 : j);
21     }
22     bool isFriend(int i, int j) {
23         _run_both;
24         return i == j && i;
25     }
26 };
27
28 typedef double elemType;
29
30 struct Edge {
31     int from, to;
32     elemType len;
33     Edge *next;
34 };
35
36 struct HeapNode {
37     int a, b;
38     elemType len;
39 };
40
41 #define _cp(a,b) ((a).len < (b).len)
42
43 class MinHeap {
44 public:
45     HeapNode h[MAXN * MAXN];
46     int n, p, c;
47     void init() {
48         n = 0;
49     }
50     void ins(HeapNode e) {
51         for (p = ++n; p > 1 && _cp(e, h[p >> 1]); h[p] = h[p >> 1], p >>= 1);
52         h[p] = e;
53     }
54     bool del(HeapNode &e) {

```

```

55     if (!n) {
56         return false;
57     }
58     e = h[p = 1];
59     for (c = 2; c < n && _cp(h[c += (c < n - 1 && _cp(h[c + 1], h[c]))], h[n]); c << \
60 = 1) {
61         h[p] = h[c];
62         p = c;
63     }
64     h[p] = h[n--];
65     return true;
66 }
67 };
68
69 elemType kruskal(int n, const Edge *list[], int edge[][2]) {
70     DSet u;
71     MinHeap h;
72     const Edge *t;
73     HeapNode e;
74     elemType ret = 0;
75     int i, m = 0;
76     u.init(), h.init();
77     for (i = 0; i < n; i++) {
78         for (t = list[i]; t; t = t->next) {
79             if (i < t->to) {
80                 e.a = i;
81                 e.b = t->to;
82                 e.len = t->len;
83                 h.ins(e);
84             }
85         }
86     }
87     while (m < n - 1 && h.del(e)) {
88         if (!u.isFriend(e.a + 1, e.b + 1)) {
89             edge[m][0] = e.a;
90             edge[m][1] = e.b;
91             ret += e.len;
92             u.setFriend(e.a + 1, e.b + 1);
93         }
94     }
95     return ret;
96 }

```

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

```

1 // 不是太苛刻的情况下推荐使用
2 // 复杂度为  $O(|E| + |V| \lg |V|)$ , 但常数较大, 而且复杂度不是很严格
3 // 边权非负!
4
5 #define Rec pair<T, int>
6
7 template<class T>
8 T Prim(int n, vector<pair<int, T> > e[], T mind[], int *pre = NULL) {
9     int s = 0;
10    priority_queue<Rec, vector<Rec>, greater<Rec> > q;

```

```

11     vector<bool> mark(n, false);
12
13     // pre 为 NULL 则不做记录
14     if (pre != NULL) {
15         fill(pre, pre + n, -1);
16         pre[s] = s;
17     }
18     // mind 初始化部分注意修改
19     fill(mind, mind + n, numeric_limits<T>::max());
20     mind[s] = T();
21     q.push(make_pair(T(), s));
22
23     T ret = T();
24
25     while (!q.empty()) {
26         s = q.top().second;
27         if (!mark[s]) {
28             mark[s] = true;
29             ret += q.top().first;
30             q.pop();
31             for (typename vector<pair<int, T> >::const_iterator i = e[s].begin(); i != e\
32 [s].end(); ++i) {
33                 if (!mark[i->first] && mind[i->first] > i->second) {
34                     mind[i->first] = i->second;
35                     if (pre != NULL) {
36                         pre[i->first] = s;
37                     }
38                     q.push(make_pair(mind[i->first], i->first));
39                 }
40             }
41             } else {
42                 q.pop();
43             }
44         }
45
46     return ret;
47 }

```

7.4.4 最小生成树 (prim 邻接阵形式)

```

1 //无向图最小生成树,prim 算法, 邻接阵形式, 复杂度  $O(n^2)$ 
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  $INF$ 
3 //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  $-1$ 
4 //必须保证图的连通的!
5 const int MAXN = 200;
6 const int INF = 1000000000;
7
8 template <class elemType>
9 elemType prim(int n, const elemType mat[][MAXN], int *pre) {
10     elemType mind[MAXN], ret = 0;
11     int v[MAXN], i, j, k;
12     for (i = 0; i < n; i++) {
13         mind[i] = INF;
14         v[i] = 0;

```

```

15     pre[i] = -1;
16 }
17 for (mind[j = 0] = 0; j < n; j++) {
18     for (k = -1, i = 0; i < n; i++) {
19         if (!v[i] && (k == -1 || mind[i] < mind[k])) {
20             k = i;
21         }
22     }
23     v[k] = 1;
24     ret += mind[k];
25     for (i = 0; i < n; i++) {
26         if (!v[i] && mat[k][i] < mind[i]) {
27             mind[i] = mat[pre[i] = k][i];
28         }
29     }
30 }
31 return ret;
32 }

```

7.4.5 次最小生成树

```

1 // 次小生成树, 复杂度  $O(n^2)$ 
2 // 传入邻接阵 mat, 不存在边权 inf
3 // 返回次小生成树长度和树的构造 pre[]
4 // 如返回 inf 则不存在次小生成树
5 // 必须保证图的连通
6 const int maxn = 100;
7 const int inf = 1000000000;
8 typedef int elem_t;
9
10 elem_t prim(int n, elem_t mat[][maxn], int *pre) {
11     elem_t min[maxn], ret = 0;
12     int v[maxn], i, j, k;
13
14     for (i = 0; i < n; ++i)
15         min[i] = inf, v[i] = 0, pre[i] = -1;
16     for (min[j = 0] = 0; j < n; ++j) {
17         for (k = -1, i = 0; i < n; ++i) if (!v[i] && (k == -1 || min[i] < min[k]))
18             k = i;
19         for (v[k] = 1, ret += min[k], i = 0; i < n; ++i)
20             if (!v[i] && mat[k][i] < min[i])
21                 min[i] = mat[pre[i] = k][i];
22     }
23
24     return ret;
25 }
26
27 elem_t sbmst(int n, elem_t mat[][maxn], int *pre) {
28     elem_t min = inf, t, ret = prim(n, mat, pre);
29     int i, j, ti, tj;
30     for (i = 0; i < n; ++i) for (j = 0; j < n && pre[i] != -1; ++j) if (i != j && pre[i] \
31 != j && pre[j] != i)
32         if (mat[j][i] < inf && (t = mat[j][i] - mat[pre[i]][i]) < min)
33             min = t, ti = i, tj = j;

```

```

34     pre[ti] = tj;
35     return ret + min;
36 }

```

7.5 网络流

7.5.1 最大流 (dinic d)

```

1  /*
2  最大流 Dinic 算法 by dd_engi
3  1. 算法被封装成了一个 struct。
4  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。
5  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 \
6  容量的边。
7  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。
8  5. 同一个 struct 可以处理多组数据，但每次都要先 init。
9  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。
10 7. 可处理多重边。
11 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 \
12 , 元素保存在传入的数组中。
13 */
14 #include <cstdio>
15 #include <cstring>
16 #include <climits>
17 using namespace std;
18
19 const int MAXN=22000,MAXM=440000;
20 struct Dinic {
21     struct edge {
22         int x,y; //两个顶点
23         int c; //容量
24         int f; //当前流量
25         edge *next,*back; //下一条边, 反向边
26         edge(int x,int y,int c,edge *next):x(x),y(y),c(c),f(0),next(next),back(0) {}
27         void *operator new(size_t, void *p) {
28             return p;
29         }
30     } *E[MAXN],*data; //E[i] 保存顶点 i 的边表
31     char storage[2*MAXM *sizeof(edge)];
32     int S,T; //源、汇
33
34     int Q[MAXN]; //DFS 用到的 queue
35     int D[MAXN]; //距离标号, -1 表示不可达
36     void DFS() {

```

```

37     memset(D, -1, sizeof(D));
38     int head=0, tail=0;
39     Q[tail++] = S;
40     D[S] = 0;
41     for(;;) {
42         int i = Q[head++];
43         for(edge *e = E[i]; e; e = e->next) {
44             if(e->c == 0) continue;
45             int j = e->y;
46             if(D[j] == -1) {
47                 D[j] = D[i] + 1;
48                 Q[tail++] = j;
49                 if(j == T) return;
50             }
51         }
52         if(head == tail) break;
53     }
54 }
55 edge *cur[MAXN]; //当前弧
56 edge *path[MAXN]; //当前找到的增广路
57 int flow() {
58     int res = 0; //结果, 即总流量
59     int path_n; //path 的大小
60     for(;;) {
61         DFS();
62         if(D[T] == -1) break;
63         memcpy(cur, E, sizeof(E));
64         path_n = 0;
65         int i = S;
66         for(;;) {
67             if(i == T) { //已找到一条增广路, 增广之
68                 int mink = 0;
69                 int delta = INT_MAX;
70                 for(int k = 0; k < path_n; ++k) {
71                     if(path[k]->c < delta) {
72                         delta = path[k]->c;
73                         mink = k;
74                     }
75                 }
76                 for(int k = 0; k < path_n; ++k) {
77                     path[k]->c -= delta;
78                     path[k]->back->c += delta;
79                 }
80                 path_n = mink; //回退
81                 i = path[path_n]->x;
82                 res += delta;
83             }
84             edge *e;
85             for(e = cur[i]; e; e = e->next) {
86                 if(e->c == 0) continue;
87                 int j = e->y;
88                 if(D[i] + 1 == D[j]) break; //找到一条弧, 加到路径里
89             }

```

```

90         cur[i]=e; //当前弧结构, 访问过的不能增广的弧不会再访问
91         if(e) {
92             path[path_n++]=e;
93             i=e->y;
94         } else { //该节点已没有任何可增广的弧, 从图中删去, 回退一步
95             D[i]=-1;
96             if(path_n==0) break;
97             path_n--;
98             i=path[path_n]->x;
99         }
100     }
101 }
102 return res;
103 }
104 int cut(int *s) {
105     int rst=0;
106     for(int i=0; i<MAXN; ++i)
107         if(D[i]==-1 && E[i])
108             s[rst++]=i;
109     return rst;
110 }
111 void init(int _S, int _T) {
112     S=_S, T=_T;
113     data=(edge *)storage;
114     memset(E, 0, sizeof(E));
115 }
116 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\
117     <w<=INT_MAX
118     E[x]=new((void *)data++) edge(x,y,w,E[x]);
119     E[y]=new((void *)data++) edge(y,x,0,E[y]);
120     E[x]->back = E[y];
121     E[y]->back = E[x];
122 }
123 };
124
125
126 /**** 用来 AC POJ3469 的示范用法 ****/
127 Dinic dinic;
128 int main() {
129     int N,M;
130     while(2==scanf("%d%d",&N,&M)) {
131         int rst=0;
132         int S=0, T=N+1;
133         dinic.init(S,T);
134         for(int i=1; i<=N; ++i) {
135             int a,b;
136             scanf("%d%d",&a,&b);
137             dinic.add_edge(S,i,a);
138             dinic.add_edge(i,T,b);
139         }
140         for(int i=0; i<M; ++i) {
141             int x,y,w;
142             scanf("%d%d%d",&x,&y,&w);
143             dinic.add_edge(x,y,w);
144             dinic.add_edge(y,x,w);

```



```

145     }
146     rst=dinic.flow();
147     printf("%d\n",rst);
148 }
149 }

```

7.5.2 最高标号先流推进

```

1 //邻接表形式, 邻接阵接口, 复杂度  $O(V^3)$ 
2 //返回最大流量, f 返回每条边的流量, 返回最大流量
3 //传入网络节点数 n, 容量 mat, 源点 s, 汇点 t
4 const int MAXN = 210;
5 const int INF = 1000000000;
6
7 int maxFlow(int n, const int mat[][MAXN], int s, int t, int f[][MAXN]) {
8     int g[MAXN][MAXN], cur[MAXN], h[MAXN], e[MAXN], q[MAXN], l[MAXN * 2][MAXN], i, j, k, \
9     head, tail, ret, checked, p, o;
10    for (i = 0; i < n; i++) {
11        h[i] = -1;
12        cur[i] = 1;
13        for (e[i] = g[i][0] = j = 0; j < n; f[i][j++] = 0) {
14            if (mat[i][j] || mat[j][i]) {
15                g[i][++g[i][0]] = j;
16            }
17        }
18    }
19    for (i = 0; i < 2 * n; i++) {
20        l[i][0] = 0;
21    }
22    for (l[h[q[head = 0] = t] = 0][++l[0][0]] = t, tail = 1; head < tail; head++) {
23        for (i = 1; i <= g[j = q[head]][0]; i++) {
24            if (h[k = g[j][i]] < 0) {
25                h[k] = h[j] + 1;
26                q[tail++] = k;
27                if (k != s) {
28                    l[h[k]][++l[h[k]][0]] = k;
29                }
30            }
31        }
32    }
33    for (h[s] = n, i = 1; i <= g[s][0]; i++) {
34        j = g[s][i];
35        f[j][s] = -(f[s][j] = e[j] = mat[s][j]);
36    }
37    for (i = n; i; i--) {
38        for (checked = 0, j = l[i][0]; j;) {
39            if ((k = l[i][j]) == s) {
40                j--;
41            } else if (!e[k]) { //Full
42                if (checked) { //Update
43                    if (i && l[i][0] == 1) {
44                        for (p = h[k] + 1; p < n; l[n][0] += l[p][0], l[p++][0] = 0) {
45                            for (o = 1; o <= l[p][0]; o++) {
46                                l[n][++l[n][0]] = l[p][o];

```

```

47         h[l[p][o]] = n;
48     }
49 }
50 }
51 l[h[k]][++l[h[k]][0]] = k;
52 l[i][j] = l[i][l[i][0]--];
53 i = h[k];
54 break;
55 } else {
56     j--;
57 }
58 } else if (cur[k] > g[k][0]) { //ReLabel
59     for (checked = p = 1, o = INF; p <= g[k][0]; p++) {
60         if (mat[k][g[k][p]] > f[k][g[k][p]] && h[g[k][p]] < o) {
61             o = h[g[k][p]];
62         }
63     }
64     h[k] = o + 1, cur[k] = 1;
65 } else if ((o = mat[k][p = g[k][cur[k]]] - f[k][p]) && h[k] == h[p] + 1) {
66     o = o < e[k] ? o : e[k];
67     f[p][k] = -(f[k][p] += o);
68     e[k] -= o;
69     e[p] += o;
70 } else {
71     cur[k]++;
72 }
73 }
74 }
75 for (ret = 0, i = 1; i <= g[s][0]; i++) {
76     ret += f[s][g[s][i]];
77 }
78 return ret;
79 }

```

7.5.3 网络流 (全功能)

```

1 // 通用网络流 (Sap + Johnson + 上下界) By 猛犸也钻地 @ 2012.02.10
2
3 #include <cstring>
4 #include <queue>
5 #include <algorithm>
6 using namespace std;
7
8 class Network {
9 public:
10     typedef int VAL; // 费用的类型
11     static const int SIZE = 1005; // 最大点数
12     static const int INF = 1000000007; // 流量的极大值
13     typedef struct ARC {
14         int t, c;
15         VAL w;
16         ARC *o;
17     } *PTR;
18     ARC arc[200005]; // 最大边数, 注意一次普通加边操作需要占用两条边

```

```

19  PTR now[SIZE],e[SIZE];          // cnt[] 为该层次下的点数, l[] 为层次标号
20  int cnt[SIZE],l[SIZE],r[SIZE],edge; // now[] 为当前弧, e[] 为出边链表
21  VAL sum;          // sum 为当前流网络下的费用
22  void clear() {
23      memset(e,edge=sum=0,sizeof(e)); // 清空边表
24  }
25  ARC &REV(PTR x) {
26      return arc[(x-arc)^1]; // 取反向边
27  }
28  // 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
29  int flow(int S, int T) {
30      return improved_sap(S,T,INF);
31  }
32  // 加入一条 x 到 y 的有向边, 容量为 c, 费用为 w
33  PTR add_edge(int x, int y, int c, VAL w = 0) {
34      e[x]=&(arc[edge++]=ARC) {
35          y,c,+w,e[x]
36      };
37      e[y]=&(arc[edge++]=ARC) {
38          x,0,-w,e[y]
39      };
40      return e[x];
41  }
42  // 加入一条 x 到 y 的无向边, 容量为 c, 费用为 0
43  PTR add_edge_simple(int x, int y, int c) {
44      e[x]=&(arc[edge++]=ARC) {
45          y,c,0,e[x]
46      };
47      e[y]=&(arc[edge++]=ARC) {
48          x,c,0,e[y]
49      };
50      return e[x];
51  }
52  // 加入一条 x 到 y 的有向边, 下界为 lo, 上界为 hi, 费用为 w
53  // 超级源在 SIZE-2, 超级汇在 SIZE-1, 注意给这两个点预留空间
54  PTR add_edge_bounded(int x, int y, int lo, int hi, VAL w = 0) {
55      add_edge(SIZE-2,y,lo,w);
56      add_edge(x,SIZE-1,lo,0);
57      return add_edge(x,y,hi-lo,w);
58  }
59  // 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
60  int aug(int S, int T, int &can) {
61      int x,z=T,use=can;
62      for(x=S; x!=T; x=now[x]->t) if(use>now[x]->c) use=now[x]->c;
63      for(x=S; x!=T; x=now[x]->t) {
64          now[x]->c-=use;
65          REV(now[x]).c+=use;
66          sum+=use*now[x]->w;
67      }
68      can-=use;
69      return z;

```

```

70 }
71 // 无权值最短路增广算法, 用在无费用的网络流上, 返回流量
72 int improved_sap(int S, int T, int can) { // can 为本次增广的流量上限
73     if(S==T) return can;
74     int in=can,x,m;
75     memcpy(now,e,sizeof(now));
76     memset(cnt,0,sizeof(cnt));
77     fill_n(l,SIZE,int(SIZE));
78     for(int i=m=l[r[0]=T]=0; i<=m; i++) {
79         cnt[l[x=r[i]]]++;
80         for(PTR u=e[x]; u; u=u->o)
81             if(l[u->t]==SIZE && REV(u).c) l[r[++m]=u->t]=l[x]+1;
82     }
83     for(x=r[S]=S; l[S]!=SIZE; x=r[x]) {
84 JMP:
85         for(PTR &u=now[x]; u; u=u->o) if(l[u->t]<l[x] && u->c) {
86             r[u->t]=x;
87             x=u->t==T?aug(S,T,can):u->t;
88             if(x==T) return in;
89             else goto JMP;
90         }
91         if(--cnt[l[x]]) break;
92         else l[x]=SIZE;
93         for(PTR u=e[x]; u; u=u->o)
94             if(l[u->t]+1<l[x] && u->c) now[x]=u,l[x]=l[u->t]+1;
95         if(l[x]!=SIZE) cnt[l[x]]++;
96     }
97     return in-can;
98 }
99 // 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
100 int spfa_johnson(int S, int T, int can) { // can 为本次增广的流量上限
101     if(S==T) return can;
102     int in=can,x,m;
103     VAL phi[SIZE],len[SIZE],MAXW=1000000007; // 费用的极大值
104     memset(l,0,sizeof(l));
105     fill_n(phi,SIZE,MAXW);
106     phi[r[0]=T]=0;
107     for(int i=m=0; i<=m; i++) { // 从汇点出发反向 SPFA
108         l[x=r[i%SIZE]]=0;
109         for(PTR u=e[x]; u; u=u->o) { // 下面这行如果是浮点比较要加 EPS
110             if(phi[x]+REV(u).w>=phi[u->t] || !REV(u).c) continue;
111             phi[u->t]=phi[x]+REV(u).w;
112             if(!l[u->t]) l[r[++m%SIZE]=u->t]=1;
113         }
114     }
115     do {
116         typedef pair<VAL,int> TPL;
117         priority_queue<TPL> q;
118         fill_n(len,SIZE,MAXW);
119         memset(l,0,sizeof(l));
120         q.push(TPL(len[T]=0,T));
121         while(q.size()) {
122             x=q.top().second;

```

```

123         q.pop();
124         if(!l[x]) l[x]=1;
125         else continue;
126         for(PTR u=e[x]; u; u=u->o) {
127             if(!REV(u).c || l[u->t]) continue;
128             VAL at=len[x]+phi[x]+REV(u).w-phi[u->t];
129             if(at>=len[u->t]) continue; // 如果是浮点比较要加 EPS
130             len[u->t]=at;
131             now[u->t]=&REV(u);
132             q.push(TPL(-at,u->t));
133         }
134     }
135     for(x=0; x<SIZE; x++) phi[x]+=len[x];
136 } while(phi[S]<MAXW && aug(S,T,can)!=T);
137 // 使用 phi[S]<MAXW 求最小费用最大流, 使用 phi[S]<0 求最小费用流
138 return in-can;
139 }
140 // 判断无源汇上下界可行流是否存在
141 // 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
142 bool feasible_bounded() {
143     flow(SIZE-2,SIZE-1);
144     for(PTR u=e[SIZE-2]; u; u=u->o) if(u->c) return false;
145     return true;
146 }
147 // 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
148 int max_flow_bounded(int S, int T) {
149     add_edge(T,S,INF);
150     bool ok=feasible_bounded();
151     int ret=e[S]->c;
152     edge-=2,e[S]=e[S]->o,e[T]=e[T]->o;
153     return ok?ret+flow(S,T):-1;
154 }
155 int min_flow_bounded(int S, int T) {
156     flow(SIZE-2,SIZE-1);
157     add_edge(T,S,INF);
158     bool ok=feasible_bounded();
159     int ret=e[S]->c;
160     edge-=2,e[S]=e[S]->o,e[T]=e[T]->o;
161     return ok?ret:-1;
162 }
163 // 将所有带下界的边合并回原图中
164 void merge_bounded() {
165     for(PTR u=e[SIZE-1]; u; u=u->o) u->c=REV(u).c=0;
166     for(PTR u=e[SIZE-2]; u; u=u->o) {
167         (u+4)->c+=u->c;
168         (u+5)->c+=REV(u).c;
169         u->c=REV(u).c=0;
170     }
171 }
172 };

```

7.6 连通性

7.6.1 支配树

```

1  /* 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
2      称为  $y$  的最近必经点, 最近必经点是唯一的, 记  $x=idom(y)$ 
3      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  $x$ ,
4      称为  $y$  的半必经点。半必经点也是唯一的, 记  $x=semi(y)$ 。
5
6      1. 设有向图  $G=(V,E)$ ,  $(G,r)$  是一个 Flow Graph, 则称  $(G,r)$  的子图
7           $D=(V, \{ (idom(i),i) \mid i \in V, i \neq r \}, r)$  为  $(G,r)$  的一棵 Dominator Tree。
8      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,
9          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。
10     3.  $x=idom(y)$ , 当且仅当有向边  $(x,y)$  是 Dominator Tree 中的一条树枝边。
11     4.  $x \in dom(y)$ , 当且仅当在 Dominator Tree 中存在一条从  $x$  到  $y$  的路径。
12     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。
13 */
14
15 /* 应用:
16     1. 求有向图的割顶: dominator tree 上的非叶节点
17     2. 有向图的必经边: 每条边上加一个点, 转化成必经点问题
18     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
19     4. 求多少个  $(x,y)$  满足存在  $1 \rightarrow x$  的路径和  $1 \rightarrow y$  的路径只有  $1$  这个公共点:
20         求出  $1$  为根的 dominator tree, 算出不合法的, 总的减去即可。
21         考虑  $1$  的每个儿子  $v$ , 同一颗子树的节点对都是非法的。
22 */
23 //succ 是原图, pred 是边反向后的图, dom 是 Dominator Tree
24 //dom 记录的是 dfs 序构成的树,  $G$  中节点  $u$  在 dom 树上的标号是  $dfn[u]$ 
25 //相反 dom 中节点  $u$  在原图  $G$  中的标号是  $pt[u]$ 
26 //调用 build 得到以  $s$  为根的 Dominator Tree
27 namespace DominatorTree {
28     int dfn[MAXN], pre[MAXN], pt[MAXN], sz;
29     int semi[MAXN], dsu[MAXN], idom[MAXN], best[MAXN];
30     int get(int x) {
31         if (x == dsu[x]) return x;
32         int y = get(dsu[x]);
33         if (semi[best[x]] > semi[best[dsu[x]]]) best[x] = best[dsu[x]];
34         return dsu[x] = y;
35     }
36     void dfs(int u, const VI succ[]) {
37         dfn[u] = sz;
38         pt[sz++] = u;
39         for (auto &v: succ[u]) if (!dfn[v]) {

```

```

40     dfs(v, succ);
41     pre[dfn[v]] = dfn[u];
42 }
43 }
44 void tarjan(const VI pred[], VI dom[]) {
45     for (int j = sz - 1, u; u = pt[j], j > 0; -- j) {
46         for (auto &tv: pred[u]) if (~dfn[tv]) {
47             int v = dfn[tv];
48             get(v);
49             if (semi[best[v]] < semi[j]) semi[j] = semi[best[v]];
50         }
51         dom[semi[j]].push_back(j);
52         int x = dsu[j] = pre[j];
53         for (auto &z: dom[x]) {
54             get(z);
55             if (semi[best[z]] < x) idom[z] = best[z];
56             else idom[z] = x;
57         }
58         dom[x].clear();
59     }
60     for (int i = 1; i < sz; ++ i) {
61         if (semi[i] != idom[i]) idom[i] = idom[idom[i]];
62         dom[idom[i]].push_back(i);
63     }
64 }
65 void build(int n, int s, const VI succ[], const VI pred[], VI dom[]) {
66     for (int i = 0; i < n; ++ i) {
67         dfn[i] = -1;
68         dom[i].clear();
69         dsu[i] = best[i] = semi[i] = i;
70     }
71     sz = 0;
72     dfs(s, succ);
73     tarjan(pred, dom);
74 }
75 }

```

7.6.2 无向图关键点、关键边和块

```

1 //无向图关键点、关键边和双连通块 (由关键点分割得到的块)
2 //复杂度  $O(m)$ , 边存在  $E$  中, 要保证传入时的  $E$  没有重边
3 //调用 ArtEdge_ArtVertex_Components(总点数)
4 //结果关键点储存在 keyV 中
5 //结果关键边储存在 keyE 中
6 #include <cstring>
7 #include <vector>
8 using namespace std;
9 #define MP(i,j) make_pair(i, j)
10 #define MAXN 10000
11 int dfn[MAXN], low[MAXN], st[MAXN], top;
12 vector <pair<int, int>> keyE;
13 vector <int> keyV, E[MAXN];
14
15 void tarjan(int now, int cnt) {

```

```

16     int part = (cnt > 1);
17     st[top++] = now;
18     dfn[now] = low[now] = cnt;
19     for (int ii = E[now].size() - 1; ii >= 0; --ii) {
20         int i = E[now][ii];
21         if (!dfn[i]) {
22             tarjan(i, cnt + 1);
23             low[now] = min(low[now], low[i]);
24             if (low[i] > dfn[now]) // 这两行用于求取关键边
25                 keyE.push_back(MP(now, i));
26
27             if (low[i] >= dfn[now]) {
28                 if(++part == 2) // 以下两行求取关键点
29                     keyV.push_back(now);
30
31                 vector<int> A; //以下四行求取双联通块
32                 A.push_back(now);
33                 for (st[top] = 0; st[top] != i; A.push_back(st[--top]));
34                 dummy(A); //每次 dummy 调用的 A 中包含一个联通块
35             }
36         } else if (dfn[i] != dfn[now] - 1)
37             low[now] = min(low[now], dfn[i]);
38     }
39     //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
40 }
41 void ArtEdge_ArtVertex_Components(int N) {
42     memset(dfn, 0, sizeof(dfn));
43     memset(low, 0, sizeof(low));
44     keyE.clear();
45     keyV.clear();
46     for (int i = top = 0; i < N; ++i)
47         if (!dfn[i])
48             tarjan(i, 1);
49 }

```

7.6.3 有向图强连通分量

```

1 //有向图强连通分量, 复杂度  $O(m)$ 
2 //边存在  $E$  中, 调用 Components(总点数)
3 //返回 id 数组储存每个点所在的强连通分量编号 (从 0 开始)
4 //最后 num 表示一共有几个强连通分量
5 #include <cstring>
6 #include <vector>
7 using namespace std;
8 #define MP(i,j) make_pair(i, j)
9 #define MAXN 10000
10 int dfn[MAXN], low[MAXN], id[MAXN], num, st[MAXN], top, in[MAXN], tot;
11 vector<int> E[MAXN];
12
13 void tarjan(int now) {
14     in[st[top++] = now] = true;
15     dfn[now] = low[now] = ++tot;

```



```

16  int i;
17  for (int ii = E[now].size() - 1; ii >= 0; --ii) {
18      i = E[now][ii];
19      if (!dfn[i]) {
20          tarjan(i);
21          low[now] = min(low[now], low[i]);
22      } else if (in[i])
23          low[now] = min(low[now], dfn[i]);
24  }
25  if (dfn[now] == low[now]) {
26      do {
27          i = st[--top];
28          in[i] = false;
29          id[i] = num;
30      } while (i != now);
31      ++num;
32  }
33 }
34 void Components(int N) {
35     memset(dfn, 0, sizeof(dfn));
36     memset(low, 0, sizeof(low));
37     memset(in, 0, sizeof(in));
38     memset(id, 0xff, sizeof(id));
39     for (int i = top = num = tot = 0; i < N; ++i)
40         if (!dfn[i])
41             tarjan(i);
42 }

```

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		$x \& (\sim(1 \ll k))$	k 从 0 开始计数
把右数第 k 位置 1		$x \mid (1 \ll k)$	k 从 0 开始计数
把右数第 k 位取反		$x \wedge (1 \ll k)$	k 从 0 开始计数
得到右数第 k 位值		$(x \gg k) \& 1$	k 从 0 开始计数
得到末尾 k 位		$x \& ((1 \ll k) - 1)$	
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	$x \mid (x + 1)$	
得到最右边的 0 的掩码	10100111 -> 00001000	$(\sim x) \& (x + 1)$	
把末尾连续 0 串置 1	01011000 -> 01011111	$x \mid (x - 1)$	如果 x 为 0 则结果为全 1

Continued on next page

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

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

```

1 // 编号 1 to n
2 // 每次第 d 个人出局
3 // 返回剩下的人的编号
4 int joseph(int n, int d) {
5     int ret = 1;
6     for (int i = 2; i <= n; i++)
7         ret = (ret + d - 1) % i + 1;
8     return ret;
9 }
10
11 //Joseph 问题模拟
12 //传入 n, m, r 返回出环的序列
13 //时间复杂度  $O(n \log n)$ 
14 #include <cstring>
15
16 const int MAXN = 32768;
17
18 void josephus(int n, int m, int r[]) {
19     int d[MAXN * 2], i, j, nbase, p, t;
20     for (nbase = 1; nbase < n; nbase <= 1);
21     memset(d, 0, sizeof(d));
22     for (i = 0; i < n; i++) {
23         d[nbase + i] = 1;
24     }

```

```

25     for (i = nbase - 1; i; i--) {
26         d[i] = d[i << 1] + d[i << 1 | 1];
27     }
28     for (j = i = 0; i < n; i++) {
29         for (j = t = (j - 1 + m) % (n - i), p = 1; p < nbase;) {
30             d[p]--;
31             if (t < d[p << 1]) {
32                 p = p << 1;
33             } else {
34                 t -= d[p << 1];
35                 p = p << 1 | 1;
36             }
37         }
38         r[i] = p - nbase;
39         d[p]--;
40     }
41 }

```

8.3 位操作

```

1 // 遍历一个掩码的所有子集掩码，不包括 0 和其自身
2 // 传入表示超集的掩码
3 void iterateSubset(int mask) {
4     for(int sub = (mask - 1) & mask; sub > 0; sub = (sub - 1) & mask) {
5         int incsub = ~sub & mask; // 递增顺序的子集
6         // gogogo
7     }
8 }
9
10 // 下一个包含同样数量二进制 1 的掩码
11 // 传入掩码，掩码不能为 0
12 unsigned snoob(unsigned mask) {
13     unsigned smallest, ripple, ones;
14     // x = xxx0 1111 0000
15     smallest = mask & -mask; // 0000 0001 0000
16     ripple = mask + smallest; // xxx1 0000 0000
17     ones = mask ^ ripple; // 0001 1111 0000
18     ones = (ones >> 2) / smallest; // 0000 0000 0111
19     return ripple | ones; // xxx1 0000 0111
20 }
21
22 // 遍历 {0, 1, ..., n-1} 的所有 k 元子集
23 // 传入 n 和 k, k 不为 0
24 void iterateSubset(int n, int k) {
25     int s = (1 << k) - 1;
26     while (!(s & 1 << n)) {
27         // gogogo

```

```

28     s = snoob(s);
29 }
30 }
31
32 // 求一个 32 位整数二进制 1 的位数
33 // 初始化函数先调用一次
34 int ones[256];
35
36 void initOnes() {
37     for (int i = 1; i < 256; ++i)
38         ones[i] = ones[i & (i - 1)] + 1;
39 }
40
41 int countOnes(int n) {
42     return ones[n & 255] + ones[(n >> 8) & 255] + ones[(n >> 16) & 255] + ones[(n >> 24) & 255];
43 }
44 }
45
46 // 求 32 位整数二进制 1 的位数, 不需要初始化 (from Beautiful Code 2007 Ch. 10)
47 int countOnes(unsigned x) {
48     x = x - ((x >> 1) & 0x55555555);
49     x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
50     x = (x + (x >> 4)) & 0x0F0F0F0F;
51     x = x + (x >> 8);
52     x = x + (x >> 16);
53     return x & 0x0000003F;
54 }
55
56
57 // 求一个 32 位整数二进制 1 的位数的奇偶性
58 // 偶数返回 0, 奇数返回 1
59 int parityOnes(unsigned n) {
60     n ^= n >> 1;
61     n ^= n >> 2;
62     n ^= n >> 4;
63     n ^= n >> 8;
64     n ^= n >> 16;
65     return n & 1; // n 的第 i 位是原数第 i 位到最左侧位的奇偶性
66 }
67
68 // 对一个 32 位整数进行位反转
69 unsigned revBits(unsigned n) {
70     n = (n & 0x55555555) << 1 | (n >> 1) & 0x55555555;
71     n = (n & 0x33333333) << 2 | (n >> 2) & 0x33333333;
72     n = (n & 0x0F0F0F0F) << 4 | (n >> 4) & 0x0F0F0F0F;
73     n = (n << 24) | ((n & 0xFF00) << 8) | ((n >> 8) & 0xFF00) | (n >> 24);
74     return n;
75 }
76
77 // 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
78 void partSum(int n, int a[], int s[]) {
79     s[0] = 0;
80     for (int i = 0; i < n; i++) {

```

```

81     s[1 << i] = a[i];
82 }
83 for (int i = 1; i < (1 << n); i++) {
84     s[i] = s[i&(i-1)] + s[i^(i&(i-1))];
85 }
86 }

```

8.4 布尔母函数

```

1 //布尔母函数
2 //判 m[] 个价值为 w[] 的货币能否构成 value
3 //适合 m[] 较大 w[] 较小的情况
4 //返回布尔量
5 //传入货币种数 n, 个数 m[], 价值 w[] 和目标值 value
6 const int MAXV = 100000;
7
8 bool genfunc(int n, const int *m, const int *w, int value) {
9     int i, j, k, c;
10    char r[MAXV];
11    for (r[0] = i = 1; i <= value; r[i++] = 0);
12    for (i = 0; i < n; i++) {
13        for (j = 0; j < w[i]; j++) {
14            c = m[i] * r[k = j];
15            while ((k += w[i]) <= value) {
16                if (r[k]) {
17                    c = m[i];
18                } else if (c) {
19                    r[k] = 1;
20                    c--;
21                }
22            }
23            if (r[value]) {
24                return true;
25            }
26        }
27    }
28    return false;
29 }

```

8.5 快速沃尔什哈达马变换

```

1 /*
2  * FWT, 默认为 xor 运算, 要修改为 and 或 or 的时候按注释改即可。
3  */
4 #include <bits/stdc++.h>
5 using namespace std;
6 #define rep(i,s,t) for (int i=s;i<=t;i++)
7 #define FOR(i,s,t) for (int i=s;i<t;i++)
8 const int maxn = 1<<12;
9 const LL mod = 1e9 + 7;
10 LL a[maxn], b[maxn];

```

```

11 LL P = mod;
12 LL PowMod(LL x, LL k) {
13     LL ret = 1;
14     for (; k >>= 1; x = x * x % mod)
15         if (k & 1) ret = (ret * x) % mod;
16     return ret;
17 }
18 LL inv2 = PowMod(2, mod - 2);
19 void TransForm(int l, int r, LL a[]) { /// [l, r)
20     if (l == r - 1) return;
21     int len = (r - l) >> 1;
22     int m = l + len;
23     TransForm(l, m, a);
24     TransForm(m, r, a);
25     FOR(i, l, m) {
26         LL a0 = a[i];
27         LL a1 = a[i + len];
28         a[i] = (a0 - a1 + mod) % mod; /* And: a[i] = a0 + a1; Or: a[i] = a0; */
29         a[i + len] = (a0 + a1) % mod; /* a[i + len] = a1; a[i + len] = a0 + a1 */
30     }
31 }
32 }
33 void uTransForm(int l, int r, LL a[]) {
34     if (l == r - 1) return;
35     int len = (r - l) >> 1;
36     int m = l + len;
37     FOR(i, l, m) {
38         LL y0 = a[i];
39         LL y1 = a[i + len];
40         a[i] = (y0 + y1) * inv2 % mod; /* And: a[i] = y0 - y1 Or: a[i] = y0 */
41         a[i + len] = (y1 - y0 + mod) * inv2 % mod; /* a[i + len] = y1 a[i + len] = y1 - y0 */
42     }
43 }
44 uTransForm(l, m, a);
45 uTransForm(m, r, a);
46 }
47 void Conv(LL a[], LL b[], int len) { /// 请保证 len 为 2 的幂次
48     TransForm(0, len, a);
49     TransForm(0, len, b);
50     FOR(i, 0, len) a[i] = a[i] * b[i] % mod;
51     uTransForm(0, len, a);
52 }
53 int main() { /// 用法如下
54     rep(i, 0, 4) a[i] = i + 1;
55     rep(i, 0, 4) b[i] = 4 - i;
56     Conv(a, b, 1 << 11);
57     rep(i, 0, 8) cout << i << " " << a[i] << endl;
58 }

```

8.6 快速线性递推

```

1 #include <stdio>
2 #include <cstring>
3 #include <algorithm>
4 using namespace std;
5
6 typedef long long LL;
7
8 const int M = 2;
9
10 // written by yxdb
11 // given first m a[i] and coef c[i] (0-based),
12 // calc a[n] mod p in O(m*m*Log(n)).
13 // a[n] = sum(c[m-i]*a[n-i]), i = 1..m
14 // i.e. a[m] = sum(c[i]*a[i]), i = 0..m-1
15 int linear_recurrence(LL n, int m, int a[], int c[], int p) {
16     LL v[M] = {1 % p}, u[M<<1], msk = !n;
17     for(LL i = n; i > 1; i >= 1) msk <= 1;
18     for(LL x = 0; msk; msk >= 1, x <= 1) {
19         fill_n(u, m<<1, 0);
20         int b = !(n & msk);
21         x |= b;
22         if(x < m) u[x] = 1 % p;
23         else {
24             for(int i = 0; i < m; ++i)
25                 for(int j = 0, t = i+b; j < m; ++j, ++t)
26                     u[t] = (u[t]+v[i]*v[j]) % p;
27             for(int i = (m<<1)-1; i >= m; --i)
28                 for(int j = 0, t = i-m; j < m; ++j, ++t)
29                     u[t] = (u[t]+c[j]*u[i]) % p;
30         }
31         copy(u, u+m, v);
32     }
33     int an = 0;
34     for(int i = 0; i < m; ++i) an = (an+v[i]*a[i]) % p;
35     return an;
36 }
37
38 const int MOD = 1000000007;
39
40 int main() {
41     int n, a[2] = {0, 1}, c[2] = {1, 1};
42     while(1==scanf("%d", &n) && n >= 0) {
43         printf("%d\n", linear_recurrence(n, 2, a, c, MOD));
44     }
45 }

```

8.7 最大子阵和

```

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

```

```

3 //返回最大子阵和, 重载返回子阵位置 (maxsum=list[s1][s2]+...+list[e1][e2])
4 //可更改元素类型
5 const int MAXN = 100;
6
7 template <class elemType>
8 elemType maxsum(int m, int n, const elemType mat[][MAXN]) {
9     elemType matsum[MAXN][MAXN + 1], ret, sum;
10    int i, j, k;
11    for (i = 0; i < m; i++) {
12        for (matsum[i][j = 0] = 0; j < n; j++) {
13            matsum[i][j + 1] = matsum[i][j] + mat[i][j];
14        }
15    }
16    for (ret = mat[0][j = 0]; j < n; j++) {
17        for (k = j; k < n; k++) {
18            for (sum = 0, i = 0; i < m; i++) {
19                sum = (sum > 0 ? sum : 0) + matsum[i][k + 1] - matsum[i][j], ret = (sum \
20 > ret ? sum : ret);
21            }
22        }
23    }
24    return ret;
25 }
26
27 template <class elemType>
28 elemType maxsum(int m, int n, const elemType mat[][MAXN], int &s1, int &s2, int &e1, int \
29 &e2) {
30    elemType matsum[MAXN][MAXN + 1], ret, sum;
31    int i, j, k, s;
32    for (i = 0; i < m; i++) {
33        for (matsum[i][j = 0] = 0; j < n; j++) {
34            matsum[i][j + 1] = matsum[i][j] + mat[i][j];
35        }
36    }
37    for (ret = mat[s1 = e1 = 0][s2 = e2 = j = 0]; j < n; j++) {
38        for (k = j; k < n; k++) {
39            for (sum = 0, s = i = 0; i < m; i++, s = (sum > 0 ? s : i)) {
40                if ((sum = (sum > 0 ? sum : 0) + matsum[i][k + 1] - matsum[i][j]) > ret) \
41 {
42                 ret = sum;
43                 s1 = s;
44                 s2 = i;
45                 e1 = j;
46                 e2 = k;
47             }
48         }
49     }
50 }
51 return ret;
52 }

```

8.8 最长公共单调子序列

```

1 /**
2  * 最长公共递增子序列, 时间复杂度  $O(n^2 * \text{Log}n)$ , 空间  $O(n^2)$ 

```



```

3  * n 为 a 的大小, m 为 b 的大小
4  * 结果在 ans 中
5  * "define _cp(a,b) ((a)<(b))" 求解最长严格递增序列
6  */
7  const int MAXN = 1000;
8  #define _cp(a,b) ((a) < (b))
9
10 typedef int elemType;
11
12 elemType DP[MAXN][MAXN];
13 int num[MAXN], p[1 << 20];
14 int lis(int n, const elemType *a, int m, const elemType *b, elemType *ans) {
15     int i, j, l, r, k;
16     DP[0][0] = 0;
17     num[0] = (b[0] == a[0]);
18     for (i = 1; i < m; i++) {
19         num[i] = (b[i] == a[0]) || num[i - 1];
20         DP[i][0] = 0;
21     }
22     for (i = 1; i < n; i++) {
23         if (b[0] == a[i] && !num[0]) {
24             num[0] = 1;
25             DP[0][0] = i << 10;
26         }
27         for (j = 1; j < m; j++) {
28             for (k = ((l = 0) + (r = num[j - 1] - 1)) >> 1; l <= r; k = (l + r) >> 1) {
29                 if (_cp(a[DP[j - 1][k] >> 10], a[i])) {
30                     l = k + 1;
31                 } else {
32                     r = k - 1;
33                 }
34             }
35             if (l < num[j - 1] && i == (DP[j - 1][l] >> 10)) {
36                 if (l >= num[j]) {
37                     DP[j][num[j]++] = DP[j - 1][l];
38                 } else {
39                     DP[j][l] = _cp(a[DP[j][l] >> 10], a[i]) ? DP[j][l] : DP[j - 1][l];
40                 }
41             }
42             if (b[j] == a[i]) {
43                 for (k = ((l = 0) + (r = num[j] - 1)) >> 1; l <= r; k = (l + r) >> 1) {
44                     if (_cp(a[DP[j][k] >> 10], a[i])) {
45                         l = k + 1;
46                     } else {
47                         r = k - 1;
48                     }
49                 }
50                 DP[j][l] = (i << 10) + j;
51                 num[j] += (l >= num[j]);
52                 p[DP[j][l]] = l ? DP[j][l - 1] : - 1;
53             }
54         }
55     }
56
57     for (k = DP[m - 1][i = num[m - 1] - 1]; i >= 0; k = p[k]) {
58         ans[i--] = a[k >> 10];
59     }
60 }

```

```

59     }
60     return num[m - 1];
61 }

```

8.9 最长子序列

```

1 //最长单调子序列, 复杂度  $O(n\log n)$ 
2 //注意最小序列覆盖和最长序列的对应关系, 例如
3 //"define _cp(a,b) ((a)>(b))" 求解最长严格递减序列, 则
4 //"define _cp(a,b) (!(a)>(b))" 求解最小严格递减序列覆盖
5 //可更改元素类型和比较函数
6 const int MAXN = 10000;
7 #define _cp(a,b) ((a)>(b))
8
9 template <class elemType>
10 int subseq(int n, const elemType *a) {
11     int b[MAXN + 1], i, l, r, m, ret = 0;
12     for (i = 0; i < n; b[l] = i++, ret += (l > ret)) {
13         for (m = ((l = 1) + (r = ret)) >> 1; l <= r; m = (l + r) >> 1) {
14             if (_cp(a[b[m]], a[i])) {
15                 l = m + 1;
16             } else {
17                 r = m - 1;
18             }
19         }
20     }
21     return ret;
22 }
23
24 template <class elemType>
25 int subseq(int n, const elemType *a, elemType *ans) {
26     int b[MAXN + 1], p[MAXN], i, l, r, m, ret = 0;
27     for (i = 0; i < n; p[b[l] = i++] = b[l - 1], ret += (l > ret)) {
28         for (m = ((l = 1) + (r = ret)) >> 1; l <= r; m = (l + r) >> 1) {
29             if (_cp(a[b[m]], a[i])) {
30                 l = m + 1;
31             } else {
32                 r = m - 1;
33             }
34         }
35     }
36     for (m = b[i = ret]; i; ans[--i] = a[m], m = p[m]);
37     return ret;
38 }

```

8.10 行列式求模

```

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

```

```

5 typedef long long LL;
6 int detMod(int n, int m, int mat[][MAXN]) {
7     for (int i = 0; i < n; i++)
8         for (int j = 0; j < n; j++)
9             mat[i][j] %= m;
10    LL ret = 1;
11    for (int i = 0; i < n; i++) {
12        for (int j = i + 1; j < n; j++)
13            while (mat[j][i] != 0) {
14                LL t = mat[i][i] / mat[j][i];
15                for (int k = i; k < n; k++) {
16                    mat[i][k] = (mat[i][k] - mat[j][k] * t) % m;
17                    int s = mat[i][k];
18                    mat[i][k] = mat[j][k];
19                    mat[j][k] = s;
20                }
21                ret = -ret;
22            }
23        if (mat[i][i] == 0)
24            return 0;
25        ret = ret * mat[i][i] % m;
26    }
27    if (ret < 0)
28        ret += m;
29    return (int)ret;
30 }
31
32 // @author Navi
33 // 高斯消元法行列式求模。复杂度  $O(n^3 + n^2 \log n)$ 。
34 //  $n$  为行列式大小, 计算  $|mat| \% m$ 
35 // 速度只比  $O(n^3 \log n)$  的快一些, 推荐用另外那个。
36 const int MAXN = 200;
37 typedef long long LL;
38 int detMod(int n, int m, int mat[][MAXN]) {
39     for (int i = 0; i < n; i++)
40         for (int j = 0; j < n; j++)
41             mat[i][j] %= m;
42    LL ret = 1;
43    for (int i = 0; i < n; i++) {
44        for (int j = i + 1; j < n; j++) {
45            LL x1 = 1, y1 = 0, x2 = 0, y2 = 1, p = i, q = j;
46            if (mat[i][i] < 0) {
47                x1 = -1;
48                mat[i][i] = -mat[i][i];
49                ret = -ret;
50            }
51            if (mat[j][i] < 0) {
52                y2 = -1;
53                mat[j][i] = -mat[j][i];
54                ret = -ret;
55            }
56            while (mat[i][i] != 0 && mat[j][i] != 0) {
57                if (mat[i][i] <= mat[j][i]) {
58                    int t = mat[j][i] / mat[i][i];
59                    mat[j][i] -= mat[i][i] * t;

```

```

60         x2 -= x1 * t;
61         y2 -= y1 * t;
62     } else {
63         int t = mat[i][i] / mat[j][i];
64         mat[i][i] -= mat[j][i] * t;
65         x1 -= x2 * t;
66         y1 -= y2 * t;
67     }
68 }
69 x1 %= m;
70 y1 %= m;
71 x2 %= m;
72 y2 %= m;
73 if (mat[i][i] == 0 && mat[j][i] != 0) {
74     ret = -ret;
75     p = j;
76     q = i;
77     mat[i][i] = mat[j][i];
78     mat[j][i] = 0;
79 }
80 for (int k = i + 1; k < n; k++) {
81     int s = mat[i][k], t = mat[j][k];
82     mat[p][k] = (s * x1 + t * y1) % m;
83     mat[q][k] = (s * x2 + t * y2) % m;
84 }
85 }
86 if (mat[i][i] == 0)
87     return 0;
88 ret = ret * mat[i][i] % m;
89 }
90 if (ret < 0)
91     ret += m;
92 return (int)ret;
93 }

```

8.11 逆序对数

```

1 //序列逆序对数, 复杂度  $O(n\log n)$ 
2 //传入序列长度和内容, 返回逆序对数
3 //可更改元素类型和比较函数
4 #include <cstring>
5 const int MAXN = 1000000;
6 #define _cp(a,b) ((a) <= (b))
7
8 typedef int elemType;
9 elemType tmp[MAXN];
10
11 long long inv(int n, elemType *a) {
12     int left = n >> 1, r = n - left, i, j;
13     long long ret = (r > 1 ? (inv(left, a) + inv(r, a + left)) : 0);
14     for (i = j = 0; i <= left; tmp[i + j] = a[i], i++) {
15         for (ret += j; j < r && (i == left || !_cp(a[i], a[left + j])); tmp[i + j] = a[1\
16 eft + j], j++);
17     }
18     memcpy(a, tmp, sizeof(elemType) * n);

```

```
19 |     return ret;
20 | }
```

Chapter 9

其他

9.1 分数

```
1  #include <cmath>
2
3  struct Frac {
4      int num, den;
5  };
6
7  int gcd(int a, int b) {
8      int t;
9      if (a < 0) {
10         a = -a;
11     }
12     if (b < 0) {
13         b = -b;
14     }
15     if (!b) {
16         return a;
17     }
18     while (t = a % b) {
19         a = b;
20         b = t;
21     }
22     return b;
23 }
24
25 void simplify(Frac &f) {
26     int t;
27     if (t = gcd(f.num, f.den)) {
28         f.num /= t;
29         f.den /= t;
30     } else {
31         f.den = 1;
32     }
33 }
34
35 Frac f(int n, int d, int s = 1) {
36     Frac ret;
37     if (d < 0) {
38         ret.num = -n;
```

```

39     ret.den = -d;
40 } else {
41     ret.num = n;
42     ret.den = d;
43 }
44 if (s) {
45     simplify(ret);
46 }
47 return ret;
48 }
49
50 Frac convert(double x) {
51     Frac ret;
52     for (ret.den = 1; fabs(x - int(x)) > 1e-10; ret.den *= 10, x *= 10);
53     ret.num = (int)x;
54     simplify(ret);
55     return ret;
56 }
57
58 int fraqcmp(Frac a, Frac b) {
59     int g1 = gcd(a.den, b.den), g2 = gcd(a.num, b.num);
60     if (!g1 || !g2) {
61         return 0;
62     }
63     return b.den / g1 * (a.num / g2) - a.den / g1 * (b.num / g2);
64 }
65
66 Frac add(Frac a, Frac b) {
67     int g1 = gcd(a.den, b.den), g2, t;
68     if (!g1) {
69         return f(1, 0, 0);
70     }
71     t = b.den / g1 * a.num + a.den / g1 * b.num;
72     g2 = gcd(g1, t);
73     return f(t / g2, a.den / g1 * (b.den / g2), 0);
74 }
75
76 Frac sub(Frac a, Frac b) {
77     return add(a, f(-b.num, b.den, 0));
78 }
79
80 Frac mul(Frac a, Frac b) {
81     int t1 = gcd(a.den, b.num), t2 = gcd(a.num, b.den);
82     if (!t1 || !t2) {
83         return f(1, 1, 0);
84     }
85     return f(a.num / t2 * (b.num / t1), a.den / t1 * (b.den / t2), 0);
86 }
87
88 Frac div(Frac a, Frac b) {
89     return mul(a, f(b.den, b.num, 0));
90 }

```

9.2 日期

```

1 int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
2

```

```

3 class Date {
4 public:
5     //判闰年
6     inline static bool isLeap(int year) {
7         return (year % 4 == 0 && year % 100 != 0) || year % 400 == 0;
8     }
9
10    int year, month, day;
11
12    //判合法性
13    inline bool isLegal() const {
14        if (month <= 0 || month > 12) {
15            return false;
16        }
17        if (month == 2) {
18            return day > 0 && day <= 28 + isLeap(year);
19        }
20        return day > 0 && day <= days[month - 1];
21    }
22
23    //比较日期大小
24    inline int compareTo(const Date &other) const {
25        if (year != other.year) {
26            return year - other.year;
27        }
28        if (month != other.month) {
29            return month - other.month;
30        }
31        return day - other.day;
32    }
33
34    //返回指定日期是星期几 0 (Sunday) ... 6 (Saturday)
35    inline int toWeekday() const {
36        int tm = month >= 3 ? (month - 2) : (month + 10);
37        int ty = month >= 3 ? year : (year - 1);
38        return (ty + ty / 4 - ty / 100 + ty / 400 + (int)(2.6 * tm - 0.2) + day) % 7;
39    }
40
41    //日期转天数偏移
42    inline int toInt() const {
43        int ret = year * 365 + (year - 1) / 4 - (year - 1) / 100 + (year - 1) / 400;
44        days[1] += isLeap(year);
45        for (int i = 0; i < month - 1; ret += days[i++]);
46        days[1] = 28;
47        return ret + day;
48    }
49
50    //天数偏移转日期
51    inline void fromInt(int a) {
52        year = a / 146097 * 400;
53        for (a %= 146097; a >= 365 + isLeap(year); a -= 365 + isLeap(year), year++);
54        days[1] += isLeap(year);
55        for (month = 1; a >= days[month - 1]; a -= days[month - 1], month++);
56        days[1] = 28;

```

```

57     day = a + 1;
58 }
59 };

```

9.3 矩阵

```

1  #include <cmath>
2  const int MAXN = 100;
3
4  #define zero(x) (fabs(x) < 1e-10)
5
6  struct mat {
7      int n, m;
8      double data[MAXN][MAXN];
9  };
10
11 bool mul(mat &c, const mat &a, const mat &b) {
12     int i, j, k;
13     if (a.m != b.n) {
14         return false;
15     }
16     c.n = a.n;
17     c.m = b.m;
18     for (i = 0; i < c.n; i++) {
19         for (j = 0; j < c.m; j++) {
20             for (k = 0; k < a.m; k++) {
21                 c.data[i][j] += a.data[i][k] * b.data[k][j];
22             }
23         }
24     }
25     return true;
26 }
27
28 bool inv(mat &a) {
29     int i, j, k, is[MAXN], js[MAXN];
30     double t;
31     if (a.n != a.m) {
32         return false;
33     }
34     for (k = 0; k < a.n; k++) {
35         for (t = 0, i = k; i < a.n; i++) {
36             for (j = k; j < a.n; j++) {
37                 if (fabs(a.data[i][j]) > t) {
38                     t = fabs(a.data[i][j]);
39                     is[k] = i; js[k] = j;
40                 }
41             }
42             if (zero(t)) {
43                 return false;
44             }
45             if (is[k] != k) {
46                 for (j = 0; j < a.n; j++) {
47                     t = a.data[k][j];
48                     a.data[k][j] = a.data[is[k]][j];
49                     a.data[is[k]][j] = t;
50                 }
51             }

```



```

52     if (js[k] != k) {
53         for (i = 0; i < a.n; i++) {
54             t = a.data[i][k];
55             a.data[i][k] = a.data[i][js[k]];
56             a.data[i][js[k]] = t;
57         }
58     }
59     a.data[k][k] = 1 / a.data[k][k];
60     for (j = 0; j < a.n; j++) {
61         if (j != k) {
62             a.data[k][j] *= a.data[k][k];
63         }
64     }
65     for (i = 0; i < a.n; i++) {
66         if (i != k) {
67             for (j = 0; j < a.n; j++) {
68                 if (j != k) {
69                     a.data[i][j] -= a.data[i][k] * a.data[k][j];
70                 }
71             }
72         }
73     }
74     for (i = 0; i < a.n; i++) {
75         if (i != k) {
76             a.data[i][k] *= -a.data[k][k];
77         }
78     }
79 }
80 for (k = a.n - 1; k >= 0; k--) {
81     for (j = 0; j < a.n; j++) {
82         if (js[k] != k) {
83             t = a.data[k][j];
84             a.data[k][j] = a.data[js[k]][j];
85             a.data[js[k]][j] = t;
86         }
87     }
88     for (i = 0; i < a.n; i++) {
89         if (is[k] != k) {
90             t = a.data[i][k];
91             a.data[i][k] = a.data[i][is[k]];
92             a.data[i][is[k]] = t;
93         }
94     }
95 }
96 return true;
97 }
98
99 double det(const mat &a) {
100     int i, j, k, sign = 0;
101     double b[MAXN][MAXN], ret = 1, t;
102     if (a.n != a.m) {
103         return 0;
104     }
105     for (i = 0; i < a.n; i++) {
106         for (j = 0; j < a.m; j++) {
107             b[i][j] = a.data[i][j];
108         }
109     }

```

```

110     for (i = 0; i < a.n; i++) {
111         if (zero(b[i][i])) {
112             for (j = i + 1; j < a.n; j++) {
113                 if (!zero(b[j][i])) {
114                     break;
115                 }
116             }
117             if (j == a.n) {
118                 return 0;
119             }
120             for (k = i; k < a.n; k++) {
121                 t = b[i][k], b[i][k] = b[j][k], b[j][k] = t;
122             }
123             sign++;
124         }
125         ret *= b[i][i];
126         for (k = i + 1; k < a.n; k++) {
127             b[i][k] /= b[i][i];
128         }
129         for (j = i + 1; j < a.n; j++) {
130             for (k = i + 1; k < a.n; k++) {
131                 b[j][k] -= b[j][i] * b[i][k];
132             }
133         }
134     }
135     if (sign & 1) {
136         ret = -ret;
137     }
138     return ret;
139 }

```

9.4 awt 基本应用

```

1 import java.io.*;
2 import java.math.*;
3 import java.util.*;
4 import java.awt.geom.*;
5
6 // 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
7 // 这些矩形的边可以不平行于坐标轴
8
9 public class Main {
10     private static double calcArea(Area area) {
11         double ret = 0;
12         PathIterator it = area.getPathIterator(null);
13         double[] old = new double[6], now = new double[2], head = null;
14         while (!it.isDone()) {
15             switch (it.currentSegment(old)) {
16                 case PathIterator.SEG_MOVETO:
17                 case PathIterator.SEG_LINETO:
18                     if (head == null) {
19                         head = new double[2];
20                         head[0] = old[0];
21                         head[1] = old[1];
22                     } else {

```

```

23         ret += now[0] * old[1] - now[1] * old[0];
24     }
25     now[0] = old[0];
26     now[1] = old[1];
27     break;
28     case PathIterator.SEG_CLOSE:
29         ret += now[0] * head[1] - now[1] * head[0];
30         head = null;
31         break;
32     }
33     it.next();
34 }
35 return ret;
36 }
37 public static void main(String[] args) {
38     Scanner sc = new Scanner(System.in);
39     Area tmpArea, combined = new Area();
40     int n = sc.nextInt();
41     double total = 0, a, b;
42     for (int i = 0; i < n; i++) {
43         Path2D.Double p = new Path2D.Double();
44         a = sc.nextDouble();
45         b = sc.nextDouble();
46         p.moveTo(a, b);
47         for (int j = 0; j < 3; j++) {
48             a = sc.nextDouble();
49             b = sc.nextDouble();
50             p.lineTo(a, b);
51         }
52         p.closePath();
53         tmpArea = new Area(p);
54         combined.add(tmpArea);
55         total += calcArea(tmpArea);
56     }
57     System.out.println(total / calcArea(combined));
58 }
59 }

```

Chapter 10

附录

10.1 应用

10.1.1 N 皇后构造解

```

1  //N 皇后构造解, n>=4
2
3  void even1(int n, int *p) {
4      int i;
5      for (i = 1; i <= n / 2; i++) {
6          p[i - 1] = 2 * i;
7      }
8      for (i = n / 2 + 1; i <= n; i++) {
9          p[i - 1] = 2 * i - n - 1;
10     }
11 }
12
13 void even2(int n, int *p) {
14     int i;
15     for (i = 1; i <= n / 2; i++) {
16         p[i - 1] = (2 * i + n / 2 - 3) % n + 1;
17     }
18     for (i = n / 2 + 1; i <= n; i++) {
19         p[i - 1] = n - (2 * (n - i + 1) + n / 2 - 3) % n;
20     }
21 }
22
23 void generate(int, int *);
24
25 void odd(int n, int *p) {
26     generate(n - 1, p);
27     p[n - 1] = n;
28 }
29
30 void generate(int n, int *p) {
31     if (n & 1) {
32         odd(n, p);
33     } else if (n % 6 != 2) {
34         even1(n, p);
35     } else {
36         even2(n, p);
37     }

```

38 }

10.1.2 大数 (整数类封装)

```

1  // 不推荐使用, 最好用 Java
2
3  #include <iostream>
4  #include <cstring>
5  using namespace std;
6
7  #define DIGIT 4
8  #define DEPTH 10000
9  #define MAX 100
10 typedef int bignum_t[MAX+1];
11
12 int read(bignum_t a, istream &is=cin) {
13     char buf[MAX*DIGIT+1], ch;
14     int i, j;
15     memset((void *)a, 0, sizeof(bignum_t));
16     if (!(is >> buf)) return 0;
17     for (a[0]=strlen(buf), i=a[0]/2-1; i>=0; i--)
18         ch=buf[i], buf[i]=buf[a[0]-1-i], buf[a[0]-1-i]=ch;
19     for (a[0]=(a[0]+DIGIT-1)/DIGIT, j=strlen(buf); j<a[0]*DIGIT; buf[j++]='0');
20     for (i=1; i<=a[0]; i++)
21         for (a[i]=0, j=0; j<DIGIT; j++)
22             a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
23     for (; !a[a[0]]&&a[a[0]]>1; a[0]--);
24     return 1;
25 }
26
27 void write(const bignum_t a, ostream &os=cout) {
28     int i, j;
29     for (os<<a[i=a[0]], i--; i; i--)
30         for (j=DEPTH/10; j; j/=10)
31             os<<a[i]/j%10;
32 }
33
34 int comp(const bignum_t a, const bignum_t b) {
35     int i;
36     if (a[0]!=b[0])
37         return a[0]-b[0];
38     for (i=a[0]; i; i--)
39         if (a[i]!=b[i])
40             return a[i]-b[i];
41     return 0;
42 }
43
44 int comp(const bignum_t a, const int b) {
45     int c[12]= {1};
46     for (c[1]=b; c[c[0]]>=DEPTH; c[c[0]+1]=c[c[0]]/DEPTH, c[c[0]]%=DEPTH, c[0]++);
47     return comp(a, c);
48 }
49
50 int comp(const bignum_t a, const int c, const int d, const bignum_t b) {
51     int i, t=0, O=-DEPTH*2;
52     if (b[0]-a[0]<d&&O)
53         return 1;

```

```

54     for (i=b[0]; i>d; i--) {
55         t=t*DEPTH+a[i-d]*c-b[i];
56         if (t>0) return 1;
57         if (t<0) return 0;
58     }
59     for (i=d; i; i--) {
60         t=t*DEPTH-b[i];
61         if (t>0) return 1;
62         if (t<0) return 0;
63     }
64     return t>0;
65 }
66
67 void add(bignum_t a,const bignum_t b) {
68     int i;
69     for (i=1; i<=b[0]; i++)
70         if ((a[i]+=b[i])>=DEPTH)
71             a[i]-=DEPTH,a[i+1]++;
72     if (b[0]>a[0])
73         a[0]=b[0];
74     else
75         for (; a[i]>=DEPTH&& i<a[0]; a[i]-=DEPTH,i++,a[i]++);
76     a[0]+=(a[a[0]+1]>0);
77 }
78
79 void add(bignum_t a,const int b) {
80     int i=1;
81     for (a[1]+=b; a[i]>=DEPTH&& i<a[0]; a[i+1]+=a[i]/DEPTH,a[i]%=DEPTH,i++);
82     for (; a[a[0]]>=DEPTH; a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
83 }
84
85 void sub(bignum_t a,const bignum_t b) {
86     int i;
87     for (i=1; i<=b[0]; i++)
88         if ((a[i]-=b[i])<0)
89             a[i+1]--,a[i]+=DEPTH;
90     for (; a[i]<0; a[i]+=DEPTH,i++,a[i]--);
91     for (; !a[a[0]]&& a[0]>1; a[0]--);
92 }
93
94 void sub(bignum_t a,const int b) {
95     int i=1;
96     for (a[1]-=b; a[i]<0; a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,\
97 i++);
98     for (; !a[a[0]]&& a[0]>1; a[0]--);
99 }
100
101 void sub(bignum_t a,const bignum_t b,const int c,const int d) {
102     int i,0=b[0]+d;
103     for (i=1+d; i<=0; i++)
104         if ((a[i]-=b[i-d]*c)<0)
105             a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH;
106     for (; a[i]<0; a[i+1]+=(a[i]-DEPTH+1)/DEPTH,a[i]-=(a[i]-DEPTH+1)/DEPTH*DEPTH,i++);
107     for (; !a[a[0]]&& a[0]>1; a[0]--);
108 }
109
110 void mul(bignum_t c,const bignum_t a,const bignum_t b) {
111     int i,j;

```

```

112     memset((void *)c,0,sizeof(bignum_t));
113     for (c[0]=a[0]+b[0]-1,i=1; i<=a[0]; i++)
114         for (j=1; j<=b[0]; j++)
115             if ((c[i+j-1]+a[i]*b[j])>=DEPTH)
116                 c[i+j]+=c[i+j-1]/DEPTH,c[i+j-1]%=DEPTH;
117     for (c[0]+=(c[c[0]+1]>0); !c[c[0]]&&c[0]>1; c[0]--);
118 }
119
120 void mul(bignum_t a,const int b) {
121     int i;
122     for (a[1]*=b,i=2; i<=a[0]; i++) {
123         a[i]*=b;
124         if (a[i-1]>=DEPTH)
125             a[i]+=a[i-1]/DEPTH,a[i-1]%=DEPTH;
126     }
127     for (; a[a[0]]>=DEPTH; a[a[0]+1]=a[a[0]]/DEPTH,a[a[0]]%=DEPTH,a[0]++);
128     for (; !a[a[0]]&&a[0]>1; a[0]--);
129 }
130
131 void mul(bignum_t b,const bignum_t a,const int c,const int d) {
132     int i;
133     memset((void *)b,0,sizeof(bignum_t));
134     for (b[0]=a[0]+d,i=d+1; i<=b[0]; i++)
135         if ((b[i]+a[i-d]*c)>=DEPTH)
136             b[i+1]+=b[i]/DEPTH,b[i]%=DEPTH;
137     for (; b[b[0]+1]; b[0]++,b[b[0]+1]=b[b[0]]/DEPTH,b[b[0]]%=DEPTH);
138     for (; !b[b[0]]&&b[0]>1; b[0]--);
139 }
140
141 void div(bignum_t c,bignum_t a,const bignum_t b) {
142     int h,l,m,i;
143     memset((void *)c,0,sizeof(bignum_t));
144     c[0]=(b[0]<a[0]+1)?(a[0]-b[0]+2):1;
145     for (i=c[0]; i; sub(a,b,c[i]=m,i-1),i--)
146         for (h=DEPTH-1,l=0,m=(h+1)>>1; h>1; m=(h+1)>>1)
147             if (comp(b,m,i-1,a)) h=m-1;
148             else l=m;
149     for (; !c[c[0]]&&c[0]>1; c[0]--);
150     c[0]=c[0]>1?c[0]:1;
151 }
152
153 void div(bignum_t a,const int b,int &c) {
154     int i;
155     for (c=0,i=a[0]; i; c=c*DEPTH+a[i],a[i]=c/b,c%=b,i--);
156     for (; !a[a[0]]&&a[0]>1; a[0]--);
157 }
158
159 void sqrt(bignum_t b,bignum_t a) {
160     int h,l,m,i;
161     memset((void *)b,0,sizeof(bignum_t));
162     for (i=b[0]=(a[0]+1)>>1; i; sub(a,b,m,i-1),b[i]+=m,i--)
163         for (h=DEPTH-1,l=0,b[i]=m=(h+1)>>1; h>1; b[i]=m=(h+1)>>1)
164             if (comp(b,m,i-1,a)) h=m-1;
165             else l=m;
166     for (; !b[b[0]]&&b[0]>1; b[0]--);
167     for (i=1; i<=b[0]; b[i++]>>=1);
168 }
169

```

```

170 int length(const bignum_t a) {
171     int t,ret;
172     for (ret=(a[0]-1)*DIGIT,t=a[a[0]]; t; t/=10,ret++);
173     return ret>0?ret:1;
174 }
175
176 int digit(const bignum_t a,const int b) {
177     int i,ret;
178     for (ret=a[(b-1)/DIGIT+1],i=(b-1)%DIGIT; i; ret/=10,i--);
179     return ret%10;
180 }
181
182 int zeronum(const bignum_t a) {
183     int ret,t;
184     for (ret=0; !a[ret+1]; ret++);
185     for (t=a[ret+1],ret*=DIGIT; !(t%10); t/=10,ret++);
186     return ret;
187 }
188
189 void comp(int *a,const int l,const int h,const int d) {
190     int i,j,t;
191     for (i=1; i<=h; i++)
192         for (t=i,j=2; t>1; j++)
193             while (!(t%j))
194                 a[j]+=d,t/=j;
195 }
196
197 void convert(int *a,const int h,bignum_t b) {
198     int i,j,t=1;
199     memset(b,0,sizeof(bignum_t));
200     for (b[0]=b[1]=1,i=2; i<=h; i++)
201         if (a[i])
202             for (j=a[i]; j; t*=i,j--)
203                 if (t*i>DEPTH)
204                     mul(b,t),t=1;
205     mul(b,t);
206 }
207
208 void combination(bignum_t a,int m,int n) {
209     int *t=new int[m+1];
210     memset((void *)t,0,sizeof(int)*(m+1));
211     comp(t,n+1,m,1);
212     comp(t,2,m-n,-1);
213     convert(t,m,a);
214     delete []t;
215 }
216
217 void permutation(bignum_t a,int m,int n) {
218     int i,t=1;
219     memset(a,0,sizeof(bignum_t));
220     a[0]=a[1]=1;
221     for (i=m-n+1; i<=m; t*=i++)
222         if (t*i>DEPTH)
223             mul(a,t),t=1;
224     mul(a,t);
225 }
226
227 #define SGN(x) ((x)>0?1:((x)<0?-1:0))

```



```

228 #define ABS(x) ((x)>0?(x):- (x))
229
230 int read(bignum_t a,int &sgn,istream &is=cin) {
231     char str[MAX*DIGIT+2],ch,*buf;
232     int i,j;
233     memset((void *)a,0,sizeof(bignum_t));
234     if (!(is>>str)) return 0;
235     buf=str,sgn=1;
236     if (*buf=='-') sgn=-1,buf++;
237     for (a[0]=strlen(buf),i=a[0]/2-1; i>=0; i--)
238         ch=buf[i],buf[i]=buf[a[0]-1-i],buf[a[0]-1-i]=ch;
239     for (a[0]=(a[0]+DIGIT-1)/DIGIT,j=strlen(buf); j<a[0]*DIGIT; buf[j++]='0');
240     for (i=1; i<=a[0]; i++)
241         for (a[i]=0,j=0; j<DIGIT; j++)
242             a[i]=a[i]*10+buf[i*DIGIT-1-j]-'0';
243     for (; !a[a[0]]&&a[a[0]]>1; a[a[0]]--);
244     if (a[a[0]]==1&&!a[1]) sgn=0;
245     return 1;
246 }
247
248 struct bignum {
249     bignum_t num;
250     int sgn;
251 public:
252     inline bignum() {
253         memset(num,0,sizeof(bignum_t));
254         num[0]=1;
255         sgn=0;
256     }
257     inline int operator!() {
258         return num[0]==1&&!num[1];
259     }
260     inline bignum &operator=(const bignum &a) {
261         memcpy(num,a.num,sizeof(bignum_t));
262         sgn=a.sgn;
263         return *this;
264     }
265     inline bignum &operator=(const int a) {
266         memset(num,0,sizeof(bignum_t));
267         num[0]=1;
268         sgn=SGN(a);
269         add(num,sgn*a);
270         return *this;
271     };
272     inline bignum &operator+=(const bignum &a) {
273         if(sgn==a.sgn)add(num,a.num);
274         else if(sgn&&a.sgn) {
275             int ret=comp(num,a.num);
276             if(ret>0)sub(num,a.num);
277             else if(ret<0) {
278                 bignum_t t;
279                 memcpy(t,num,sizeof(bignum_t));
280                 memcpy(num,a.num,sizeof(bignum_t));
281                 sub(num,t);
282                 sgn=a.sgn;
283             } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
284         } else if(!sgn)memcpy(num,a.num,sizeof(bignum_t)),sgn=a.sgn;
285         return *this;

```

```

286     }
287     inline bignum &operator+=(const int a) {
288         if(sgn*a>0)add(num,ABS(a));
289         else if(sgn&& a) {
290             int ret=comp(num,ABS(a));
291             if(ret>0)sub(num,ABS(a));
292             else if(ret<0) {
293                 bignum_t t;
294                 memcpy(t,num,sizeof(bignum_t));
295                 memset(num,0,sizeof(bignum_t));
296                 num[0]=1;
297                 add(num,ABS(a));
298                 sgn=-sgn;
299                 sub(num,t);
300             } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
301         } else if(!sgn)sgn=SGN(a),add(num,ABS(a));
302         return *this;
303     }
304     inline bignum operator+(const bignum &a) {
305         bignum ret;
306         memcpy(ret.num,num,sizeof(bignum_t));
307         ret.sgn=sgn;
308         ret+=a;
309         return ret;
310     }
311     inline bignum operator+(const int a) {
312         bignum ret;
313         memcpy(ret.num,num,sizeof(bignum_t));
314         ret.sgn=sgn;
315         ret+=a;
316         return ret;
317     }
318     inline bignum &operator--=(const bignum &a) {
319         if(sgn*a.sgn<0)add(num,a.num);
320         else if(sgn&&a.sgn) {
321             int ret=comp(num,a.num);
322             if(ret>0)sub(num,a.num);
323             else if(ret<0) {
324                 bignum_t t;
325                 memcpy(t,num,sizeof(bignum_t));
326                 memcpy(num,a.num,sizeof(bignum_t));
327                 sub(num,t);
328                 sgn=-sgn;
329             } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
330         } else if(!sgn)add(num,a.num),sgn=-a.sgn;
331         return *this;
332     }
333     inline bignum &operator--=(const int a) {
334         if(sgn*a<0)add(num,ABS(a));
335         else if(sgn&&a) {
336             int ret=comp(num,ABS(a));
337             if(ret>0)sub(num,ABS(a));
338             else if(ret<0) {
339                 bignum_t t;
340                 memcpy(t,num,sizeof(bignum_t));
341                 memset(num,0,sizeof(bignum_t));
342                 num[0]=1;
343                 add(num,ABS(a));

```

```

344         sub(num,t);
345         sgn=-sgn;
346     } else memset(num,0,sizeof(bignum_t)),num[0]=1,sgn=0;
347 } else if(!sgn)sgn=-SGN(a),add(num,ABS(a));
348 return *this;
349 }
350 inline bignum operator-(const bignum &a) {
351     bignum ret;
352     memcpy(ret.num,num,sizeof(bignum_t));
353     ret.sgn=sgn;
354     ret-=a;
355     return ret;
356 }
357 inline bignum operator-(const int a) {
358     bignum ret;
359     memcpy(ret.num,num,sizeof(bignum_t));
360     ret.sgn=sgn;
361     ret-=a;
362     return ret;
363 }
364 inline bignum &operator*=(const bignum &a) {
365     bignum_t t;
366     mul(t,num,a.num);
367     memcpy(num,t,sizeof(bignum_t));
368     sgn*=a.sgn;
369     return *this;
370 }
371 inline bignum &operator*=(const int a) {
372     mul(num,ABS(a));
373     sgn*=SGN(a);
374     return *this;
375 }
376 inline bignum operator*(const bignum &a) {
377     bignum ret;
378     mul(ret.num,num,a.num);
379     ret.sgn=sgn*a.sgn;
380     return ret;
381 }
382 inline bignum operator*(const int a) {
383     bignum ret;
384     memcpy(ret.num,num,sizeof(bignum_t));
385     mul(ret.num,ABS(a));
386     ret.sgn=sgn*SGN(a);
387     return ret;
388 }
389 inline bignum &operator/=(const bignum &a) {
390     bignum_t t;
391     div(t,num,a.num);
392     memcpy(num,t,sizeof(bignum_t));
393     sgn=(num[0]==1&&!num[1])?0:sgn*a.sgn;
394     return *this;
395 }
396 inline bignum &operator/=(const int a) {
397     int t;
398     div(num,ABS(a),t);
399     sgn=(num[0]==1&&!num[1])?0:sgn*SGN(a);
400     return *this;
401 }

```

```

402 inline bignum operator/(const bignum &a) {
403     bignum ret;
404     bignum_t t;
405     memcpy(t,num,sizeof(bignum_t));
406     div(ret.num,t,a.num);
407     ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*a.sgn;
408     return ret;
409 }
410 inline bignum operator/(const int a) {
411     bignum ret;
412     int t;
413     memcpy(ret.num,num,sizeof(bignum_t));
414     div(ret.num,ABS(a),t);
415     ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn*SGN(a);
416     return ret;
417 }
418 inline bignum &operator%=(const bignum &a) {
419     bignum_t t;
420     div(t,num,a.num);
421     if (num[0]==1&&!num[1])sgn=0;
422     return *this;
423 }
424 inline int operator%=(const int a) {
425     int t;
426     div(num,ABS(a),t);
427     memset(num,0,sizeof(bignum_t));
428     num[0]=1;
429     add(num,t);
430     return t;
431 }
432 inline bignum operator%(const bignum &a) {
433     bignum ret;
434     bignum_t t;
435     memcpy(ret.num,num,sizeof(bignum_t));
436     div(t,ret.num,a.num);
437     ret.sgn=(ret.num[0]==1&&!ret.num[1])?0:sgn;
438     return ret;
439 }
440 inline int operator%(const int a) {
441     bignum ret;
442     int t;
443     memcpy(ret.num,num,sizeof(bignum_t));
444     div(ret.num,ABS(a),t);
445     memset(ret.num,0,sizeof(bignum_t));
446     ret.num[0]=1;
447     add(ret.num,t);
448     return t;
449 }
450 inline bignum &operator++() {
451     *this+=1;
452     return *this;
453 }
454 inline bignum &operator--() {
455     *this-=1;
456     return *this;
457 };
458 inline int operator>(const bignum &a) {
459     return sgn>0?(a.sgn>0?comp(num,a.num)>0:1):(sgn<0?(a.sgn<0?comp(num,a.num)<0:0):\

```

```

460 a.sgn<0);
461 }
462 inline int operator>(const int a) {
463     return sgn>0?(a>0?comp(num,a)>0:1):(sgn<0?(a<0?comp(num,-a)<0:0):a<0);
464 }
465 inline int operator>=(const bignum &a) {
466     return sgn>0?(a.sgn>0?comp(num,a.num)>=0:1):(sgn<0?(a.sgn<0?comp(num,a.num)<=0:0)\
467 ):a.sgn<=0);
468 }
469 inline int operator>=(const int a) {
470     return sgn>0?(a>0?comp(num,a)>=0:1):(sgn<0?(a<0?comp(num,-a)<=0:0):a<=0);
471 }
472 inline int operator<(const bignum &a) {
473     return sgn<0?(a.sgn<0?comp(num,a.num)>0:1):(sgn>0?(a.sgn>0?comp(num,a.num)<0:0):\
474 a.sgn>0);
475 }
476 inline int operator<(const int a) {
477     return sgn<0?(a<0?comp(num,-a)>0:1):(sgn>0?(a>0?comp(num,a)<0:0):a>0);
478 }
479 inline int operator<=(const bignum &a) {
480     return sgn<0?(a.sgn<0?comp(num,a.num)>=0:1):(sgn>0?(a.sgn>0?comp(num,a.num)<=0:0)\
481 ):a.sgn>=0);
482 }
483 inline int operator<=(const int a) {
484     return sgn<0?(a<0?comp(num,-a)>=0:1):(sgn>0?(a>0?comp(num,a)<=0:0):a>=0);
485 }
486 inline int operator==(const bignum &a) {
487     return (sgn==a.sgn)?!comp(num,a.num):0;
488 }
489 inline int operator==(const int a) {
490     return (sgn*a>=0)?!comp(num,ABS(a)):0;
491 }
492 inline int operator!=(const bignum &a) {
493     return (sgn==a.sgn)?comp(num,a.num):1;
494 }
495 inline int operator!=(const int a) {
496     return (sgn*a>=0)?comp(num,ABS(a)):1;
497 }
498 inline int operator[](const int a) {
499     return digit(num,a);
500 }
501 friend inline istream &operator>>(istream &is,bignum &a) {
502     read(a.num,a.sgn,is);
503     return is;
504 }
505 friend inline ostream &operator<<(ostream &os,const bignum &a) {
506     if(a.sgn<0)os<<"-";
507     write(a.num,os);
508     return os;
509 }
510 friend inline bignum sqrt(const bignum &a) {
511     bignum ret;
512     bignum_t t;
513     memcpy(t,a.num,sizeof(bignum_t));
514     sqrt(ret.num,t);
515     ret.sgn=ret.num[0]!=1|ret.num[1];
516     return ret;
517 }

```

```

518     friend inline bignum sqrt(const bignum &a,bignum &b) {
519         bignum ret;
520         memcpy(b.num,a.num,sizeof(bignum_t));
521         sqrt(ret.num,b.num);
522         ret.sgn=ret.num[0]!=1||ret.num[1];
523         b.sgn=b.num[0]!=1||ret.num[1];
524         return ret;
525     }
526     inline int length() {
527         return ::length(num);
528     }
529     inline int zeronum() {
530         return ::zeronum(num);
531     }
532     inline bignum C(const int m,const int n) {
533         combination(num,m,n);
534         sgn=1;
535         return *this;
536     }
537     inline bignum P(const int m,const int n) {
538         permutation(num,m,n);
539         sgn=1;
540         return *this;
541     }
542 };

```

10.1.3 幻方构造

```

1  //幻方构造 (L!=2)
2  const int MAXN = 100;
3
4  void dllb(int L, int si, int sj, int sn, int d[][MAXN]) {
5      int n, i = 0, j = L / 2;
6      for (n = 1; n <= L * L; n++) {
7          d[i + si][j + sj] = n + sn;
8          if (n % L) {
9              i = (i) ? (i - 1) : (L - 1);
10             j = (j == L - 1) ? 0 : (j + 1);
11         } else {
12             i = (i == L - 1) ? 0 : (i + 1);
13         }
14     }
15 }
16
17 void magicOdd(int L, int d[][MAXN]) {
18     dllb(L, 0, 0, 0, d);
19 }
20
21 void magic4k(int L, int d[][MAXN]) {
22     int i, j;
23     for (i = 0; i < L; i++) {
24         for (j = 0; j < L; j++) {
25             d[i][j] = ((i % 4 == 0 || i % 4 == 3) && (j % 4 == 0 || j % 4 == 3) || (i % \
26 4 == 1 || i % 4 == 2) && (j % 4 == 1 || j % 4 == 2)) ? (L * L - (i * L + j)) : (i * L + \
27 j + 1);
28         }
29     }

```

```

30 }
31
32 void magicOther(int L, int d[][MAXN]) {
33     int i, j, t;
34     dllb(L / 2, 0, 0, 0, d);
35     dllb(L / 2, L / 2, L / 2, L * L / 4, d);
36     dllb(L / 2, 0, L / 2, L * L / 2, d);
37     dllb(L / 2, L / 2, 0, L * L / 4 * 3, d);
38     for (i = 0; i < L / 2; i++) {
39         for (j = 0; j < L / 4; j++) {
40             if (i != L / 4 || j) {
41                 t = d[i][j];
42                 d[i][j] = d[i + L / 2][j];
43                 d[i + L / 2][j] = t;
44             }
45         }
46     }
47     t = d[L / 4][L / 4];
48     d[L / 4][L / 4] = d[L / 4 + L / 2][L / 4];
49     d[L / 4 + L / 2][L / 4] = t;
50     for (i = 0; i < L / 2; i++) {
51         for (j = L - L / 4 + 1; j < L; j++) {
52             t = d[i][j];
53             d[i][j] = d[i + L / 2][j];
54             d[i + L / 2][j] = t;
55         }
56     }
57 }
58
59 void generate(int L, int d[][MAXN]) {
60     if (L % 2) {
61         magicOdd(L, d);
62     } else if (L % 4 == 0) {
63         magic4k(L, d);
64     } else {
65         magicOther(L, d);
66     }
67 }

```

10.1.4 最大子串匹配

```

1 //最大子串匹配, 复杂度  $O(mn)$ 
2 //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
3 //注意做字符串匹配是串末的 '\0' 没有置!
4 //可更改元素类型, 更换匹配函数和匹配价值函数
5 #include <cstring>
6 #include <algorithm>
7 using namespace std;
8
9 const int MAXN = 100;
10
11 #define _match(a, b) ((a) == (b))
12 #define _value(a, b) 1
13
14 template <class elemType>

```

```

15 int strMatch(int m, const elemType *a, int n, const elemType *b) {
16     int match[MAXN + 1][MAXN + 1], i, j;
17     memset(match, 0, sizeof(match));
18     for (i = 0; i < m; i++) {
19         for (j = 0; j < n; j++) {
20             match[i + 1][j + 1] = max(max(match[i][j + 1], match[i + 1][j]), (match[i][j] \
21 ] + _value(a[i], b[i])) * _match(a[i], b[j]));
22         }
23     }
24     return match[m][n];
25 }
26
27 template <class elemType>
28 int strMatch(int m, const elemType *a, int n, const elemType *b, elemType *ret) {
29     int match[MAXN + 1][MAXN + 1], last[MAXN + 1][MAXN + 1], i, j, t;
30     memset(match, 0, sizeof(match));
31     for (i = 0; i < m; i++) {
32         for (j = 0; j < n; j++) {
33             match[i + 1][j + 1] = (match[i][j + 1] > match[i + 1][j] ? match[i][j + 1] : \
34 match[i + 1][j]);
35             last[i + 1][j + 1] = (match[i][j + 1] > match[i + 1][j] ? 3 : 1);
36             if ((t = (match[i][j] + _value(a[i], b[i])) * _match(a[i], b[j])) > match[i + \
37 1][j + 1]) {
38                 match[i + 1][j + 1] = t;
39                 last[i + 1][j + 1] = 2;
40             }
41         }
42     }
43     for (; match[i][j]; i -= (last[t = i][j] > 1), j -= (last[t][j] < 3)) {
44         ret[match[i][j] - 1] = (last[i][j] < 3 ? a[i - 1] : b[j - 1]);
45     }
46     return match[m][n];
47 }

```

10.1.5 最大子段和

```

1 //求最大子段和, 复杂度 O(n)
2 //传入串长 n 和内容 list[]
3 //返回最大子段和, 重载返回子段位置 (maxsum=list[start]+...+list[end])
4 //可更改元素类型
5 template <class elemType>
6 elemType maxsum(int n, const elemType *list) {
7     elemType ret, sum = 0;
8     int i;
9     for (ret = list[i = 0]; i < n; i++) {
10         sum = (sum > 0 ? sum : 0) + list[i];
11         ret = (sum > ret ? sum : ret);
12     }
13     return ret;
14 }
15
16 template <class elemType>
17 elemType maxsum(int n, const elemType *list, int &start, int &end) {
18     elemType ret, sum = 0;
19     int s, i;

```



```

20     for (ret = list[start = end = s = i = 0]; i < n; i++, s = (sum > 0 ? s : i)) {
21         if ((sum = (sum > 0 ? sum : 0) + list[i]) > ret) {
22             ret = sum;
23             start = s;
24             end = i;
25         }
26     }
27     return ret;
28 }

```

10.1.6 第 k 元素

```

1  //一般可用 STL 的 kth_element()
2
3  //取第 k 个元素,k=0..n-1
4  //平均复杂度 O(n)
5  //注意 a[] 中的顺序被改变
6  #define _cp(a, b) ((a) < (b))
7
8  template <class elemType>
9  elemType kthElement(int n, const elemType *a, int k) {
10     elemType t, key;
11     int left = 0, r = n - 1, i, j;
12     while (left < r) {
13         for (key = a[((i = left - 1) + (j = r + 1)) >> 1]; i < j;) {
14             for (j--; _cp(key, a[j]); j--);
15             for (i++; _cp(a[i], key); i++);
16             if (i < j) {
17                 t = a[i];
18                 a[i] = a[j];
19                 a[j] = t;
20             }
21         }
22         if (k > j) {
23             left = j + 1;
24         } else {
25             r = j;
26         }
27     }
28     return a[k];
29 }

```

10.1.7 骰子

```

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

```

```

8 //通过上表面和前面，得出右侧面
9 const int getFace[7][7]= //[upward][forward]
10     //0 1 2 3 4 5 6
11 { {0,0,0,0,0,0,0} //0
12   ,{0,0,3,5,2,4,0} //1
13   ,{0,4,0,1,6,0,3} //2
14   ,{0,2,6,0,0,1,5} //3
15   ,{0,5,1,0,0,6,2} //4
16   ,{0,3,0,6,1,0,4} //5
17   ,{0,0,4,2,5,3,0}
18 }; //6
19 //对面对应的标号
20 const int oppo[7]= {0,6,5,4,3,2,1};
21
22 class Dice {
23 public:
24     int up,forward,right;
25
26     Dice() {}
27
28     Dice(int a,int b):up(a),forward(b) {
29         right=getFace[a][b];
30     }
31
32     Dice(int a,int b,int c):up(a),forward(b),right(c) {}
33
34     void goLeft() {
35         int a=up,b=forward,c=right;
36         up=c;
37         forward=b;
38         righth=oppo[a];
39     }
40
41     void goRight() {
42         int a=up,b=forward,c=right;
43         up=oppo[c];
44         forward=b;
45         righth=a;
46     }
47
48     void goUp() {
49         int a=up,b=forward,c=right;
50         up=b;
51         forward=oppo[a];
52         righth=c;
53     }
54
55     void goDown(Pos &tmppos,Pos &pos) {
56         int a=up,b=forward,c=right;
57         up=oppo[b];
58         forward=a;
59         righth=c;

```

```

60     }
61
62 };

```

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：用最少数量的团覆盖所有的点
推论 -> 团数 \leq 色数，最大独立集数 \leq 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 $C_n(n > 3)$
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点

// 重点内容

12. 完美消除序列：点的序列 v_1, v_2, \dots, v_n ，满足 v_i 在 $\{v_i, v_{i+1}, \dots, v_n\}$ 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 v_i ，其出边为 $v_{i1}, v_{i2}, \dots, v_{ik}$
然后判断 v_{i1} 与 $v_{i2}, v_{i3}, \dots, v_{ik}$ 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 $\{p_1, p_2, \dots, p_k\}$
则 $\{N(p_1) + \{p_1\}, N(p_2) + \{p_2\}, \dots, N(p_k) + \{p_k\}\}$ 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 v_0 连接的边不能超过 K 个或只能刚好 K 个
1. 去掉所有和 v_0 连接的边，对每个连通分量求最小生成树
 2. 如果除去点 v_0 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 v_0 的路径上权值最大的边是多少，再枚举和 v_0 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 v_0 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 $O(\log N)$ 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 $dp[1 \ll i][A_i]$ 被初始化为 0, A_i 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
 $dp[mask][x] = \min(dp[mask][x], dp[sub][x] + dp[mask \setminus sub][x])$
 - 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 $F_k(v) = \infty$ 。

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

$F_k(v)$ 可以通过 $O(VE)$ 的递推方法求出。

Chapter 11

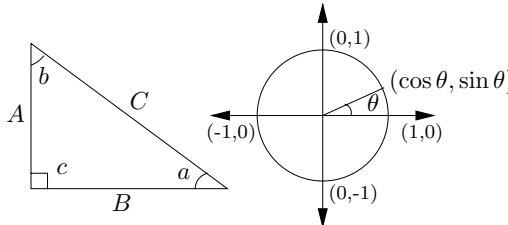
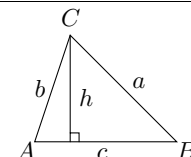
Cheat Sheet

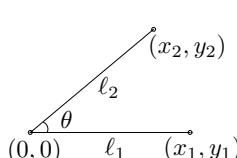
11.1 Theoretical Computer Science

Theoretical Computer Science Cheat Sheet		
Definitions		Series
$f(n) = O(g(n))$	iff \exists positive c, n_0 such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$.	$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$
$f(n) = \Omega(g(n))$	iff \exists positive c, n_0 such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$.	In general:
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.	$\sum_{i=1}^n i^m = \frac{1}{m+1} \left[(n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$
$f(n) = o(g(n))$	iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$.	$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$
$\lim_{n \rightarrow \infty} a_n = a$	iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a < \epsilon, \forall n \geq n_0$.	Geometric series:
$\sup S$	least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$.	$\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad c < 1,$
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$.	$\sum_{i=0}^n ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad c < 1.$
$\liminf_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	Harmonic series:
$\limsup_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	$H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.	$\sum_{i=1}^n H_i = (n+1)H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1} \right).$
$[n_k]$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.	1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$
$\{n_k\}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\langle n_k \rangle$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with k ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\langle\langle n_k \rangle\rangle$	2nd order Eulerian numbers.	8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.	10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{matrix} n \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} n \\ n \end{matrix} \right\} = 1,$
14. $\left[\begin{matrix} n \\ 1 \end{matrix} \right] = (n-1)!,$	15. $\left[\begin{matrix} n \\ 2 \end{matrix} \right] = (n-1)!H_{n-1},$	12. $\left\{ \begin{matrix} n \\ 2 \end{matrix} \right\} = 2^{n-1} - 1, \quad 13. \left\{ \begin{matrix} n \\ k \end{matrix} \right\} = k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\} + \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\},$
16. $\left[\begin{matrix} n \\ n \end{matrix} \right] = 1,$	17. $\left[\begin{matrix} n \\ k \end{matrix} \right] \geq \left\{ \begin{matrix} n \\ k \end{matrix} \right\},$	
18. $\left[\begin{matrix} n \\ k \end{matrix} \right] = (n-1) \left[\begin{matrix} n-1 \\ k \end{matrix} \right] + \left[\begin{matrix} n-1 \\ k-1 \end{matrix} \right],$	19. $\left\{ \begin{matrix} n \\ n-1 \end{matrix} \right\} = \left[\begin{matrix} n \\ n-1 \end{matrix} \right] = \binom{n}{2},$	20. $\sum_{k=0}^n \left[\begin{matrix} n \\ k \end{matrix} \right] = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$
22. $\langle \begin{matrix} n \\ 0 \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1 \end{matrix} \rangle = 1,$	23. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1-k \end{matrix} \rangle,$	24. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = (k+1) \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle + (n-k) \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle,$
25. $\langle \begin{matrix} 0 \\ k \end{matrix} \rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$	26. $\langle \begin{matrix} n \\ 1 \end{matrix} \rangle = 2^n - n - 1,$	27. $\langle \begin{matrix} n \\ 2 \end{matrix} \rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$
28. $x^n = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{x+k}{n},$	29. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$	30. $m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{k}{n-m},$
31. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$	32. $\langle\langle \begin{matrix} n \\ 0 \end{matrix} \rangle\rangle = 1,$	33. $\langle\langle \begin{matrix} n \\ n \end{matrix} \rangle\rangle = 0 \quad \text{for } n \neq 0,$
34. $\langle\langle \begin{matrix} n \\ k \end{matrix} \rangle\rangle = (k+1) \langle\langle \begin{matrix} n-1 \\ k \end{matrix} \rangle\rangle + (2n-1-k) \langle\langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle\rangle,$	35. $\sum_{k=0}^n \langle\langle \begin{matrix} n \\ k \end{matrix} \rangle\rangle = \frac{(2n)^n}{2^n},$	
36. $\left\{ \begin{matrix} x \\ x-n \end{matrix} \right\} = \sum_{k=0}^n \langle\langle \begin{matrix} n \\ k \end{matrix} \rangle\rangle \binom{x+n-1-k}{2n},$	37. $\left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} = \sum_{k=0}^n \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (m+1)^{n-k},$	

Theoretical Computer Science Cheat Sheet		
Identities Cont.		Trees
<p>38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix},$</p> <p>40. $\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} (-1)^{n-k},$</p> <p>42. $\left\{ \begin{matrix} m+n+1 \\ m \end{matrix} \right\} = \sum_{k=0}^m k \left\{ \begin{matrix} n+k \\ k \end{matrix} \right\},$</p> <p>44. $\binom{n}{m} = \sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k},$</p> <p>46. $\left\{ \begin{matrix} n \\ n-m \end{matrix} \right\} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix},$</p> <p>48. $\left\{ \begin{matrix} n \\ \ell+m \end{matrix} \right\} \binom{\ell+m}{\ell} = \sum_k \left\{ \begin{matrix} k \\ \ell \end{matrix} \right\} \left\{ \begin{matrix} n-k \\ m \end{matrix} \right\} \binom{n}{k},$</p>	<p>39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^n \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \binom{x+k}{2n},$</p> <p>41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k},$</p> <p>43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^m k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix},$</p> <p>45. $(n-m)! \binom{n}{m} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (-1)^{m-k}, \text{ for } n \geq m,$</p> <p>47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\},$</p> <p>49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}.$</p>	<p>Every tree with n vertices has $n-1$ edges.</p> <p>Kraft inequality: If the depths of the leaves of a binary tree are d_1, \dots, d_n:</p> $\sum_{i=1}^n 2^{-d_i} \leq 1,$ <p>and equality holds only if every internal node has 2 sons.</p>
Recurrences		
<p>Master method:</p> $T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1$ <p>If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then</p> $T(n) = \Theta(n^{\log_b a}).$ <p>If $f(n) = \Theta(n^{\log_b a})$ then</p> $T(n) = \Theta(n^{\log_b a} \log_2 n).$ <p>If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then</p> $T(n) = \Theta(f(n)).$ <p>Substitution (example): Consider the following recurrence</p> $T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$ <p>Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have</p> $t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$ <p>Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get</p> $\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$ <p>Substituting we find</p> $u_{i+1} = \frac{1}{2} + u_i, \quad u_1 = \frac{1}{2},$ <p>which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.</p> <p>Summing factors (example): Consider the following recurrence</p> $T(n) = 3T(n/2) + n, \quad T(1) = 1.$ <p>Rewrite so that all terms involving T are on the left side</p> $T(n) - 3T(n/2) = n.$ <p>Now expand the recurrence, and choose a factor which makes the left side “telescope”</p>	$1(T(n) - 3T(n/2)) = n$ $3(T(n/2) - 3T(n/4)) = n/2$ $\vdots \quad \vdots \quad \vdots$ $3^{\log_2 n - 1} (T(2) - 3T(1)) = 2$ <p>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</p> $\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$ <p>Let $c = \frac{3}{2}$. Then we have</p> $n \sum_{i=0}^{m-1} c^i = n \left(\frac{c^m - 1}{c - 1} \right)$ $= 2n(c^{\log_2 n} - 1)$ $= 2n(c^{(\log_2 n - 1) \log_2 c} - 1)$ $= 2n^k - 2n,$ <p>and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider</p> $T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$ <p>Note that</p> $T_{i+1} = 1 + \sum_{j=0}^i T_j.$ <p>Subtracting we find</p> $T_{i+1} - T_i = 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j$ $= T_i.$ <p>And so $T_{i+1} = 2T_i = 2^{i+1}$.</p>	<p>Generating functions:</p> <ol style="list-style-type: none"> 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. <p>Example:</p> $g_{i+1} = 2g_i + 1, \quad g_0 = 0.$ <p>Multiply and sum:</p> $\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$ <p>We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:</p> $\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$ <p>Simplify:</p> $\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$ <p>Solve for $G(x)$:</p> $G(x) = \frac{x}{(1-x)(1-2x)}.$ <p>Expand this using partial fractions:</p> $G(x) = x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right)$ $= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right)$ $= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}.$ <p>So $g_i = 2^i - 1$.</p>

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	General	Probability
1	2	2	Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$): $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},$ $B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$	Continuous distributions: If
2	4	3		$\Pr[a < X < b] = \int_a^b p(x) dx,$
3	8	5		then p is the probability density function of X . If
4	16	7	Change of base, quadratic formula:	$\Pr[X < a] = P(a),$
5	32	11	$\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$	then P is the distribution function of X . If P and p both exist then
6	64	13	Euler's number e :	$P(a) = \int_{-\infty}^a p(x) dx.$
7	128	17	$e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$	Expectation: If X is discrete
8	256	19	$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$	$E[g(X)] = \sum_x g(x) \Pr[X = x].$
9	512	23	$\left(1 + \frac{1}{n}\right)^n < e < \left(1 + \frac{1}{n}\right)^{n+1}.$	If X continuous then
10	1,024	29	$\left(1 + \frac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$	$E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$
11	2,048	31	Harmonic numbers:	Variance, standard deviation:
12	4,096	37	$1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \dots$	$\text{VAR}[X] = E[X^2] - E[X]^2,$
13	8,192	41	$\ln n < H_n < \ln n + 1,$	$\sigma = \sqrt{\text{VAR}[X]}.$
14	16,384	43	$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$	For events A and B :
15	32,768	47	Factorial, Stirling's approximation:	$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$
16	65,536	53	$1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$	$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$
17	131,072	59	$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$	iff A and B are independent.
18	262,144	61	Ackermann's function and inverse:	$\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$
19	524,288	67	$a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$	For random variables X and Y :
20	1,048,576	71	$\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$	$E[X \cdot Y] = E[X] \cdot E[Y],$
21	2,097,152	73		if X and Y are independent.
22	4,194,304	79	Binomial distribution:	$E[X + Y] = E[X] + E[Y],$
23	8,388,608	83	$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$	$E[cX] = cE[X].$
24	16,777,216	89	$E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$	Bayes' theorem:
25	33,554,432	97	Poisson distribution:	$\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$
26	67,108,864	101	$\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.$	Inclusion-exclusion:
27	134,217,728	103	Normal (Gaussian) distribution:	$\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] +$
28	268,435,456	107	$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.$	$\sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$
29	536,870,912	109	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	Moment inequalities:
30	1,073,741,824	113	$nH_n.$	$\Pr[X \geq \lambda E[X]] \leq \frac{1}{\lambda},$
31	2,147,483,648	127		$\Pr[X - E[X] \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$
32	4,294,967,296	131		Geometric distribution:
Pascal's Triangle				$\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$
1				$E[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$
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				

Theoretical Computer Science Cheat Sheet																											
Trigonometry	Matrices	More Trig.																									
<div></div> <p>Pythagorean theorem: $C^2 = A^2 + B^2$.</p> <p>Definitions:</p> $\sin a = A/C, \quad \cos a = B/C,$ $\csc a = C/A, \quad \sec a = C/B,$ $\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$ <p>Area, radius of inscribed circle: $\frac{1}{2}AB, \quad \frac{AB}{A+B+C}.$</p> <p>Identities:</p> $\sin x = \frac{1}{\csc x}, \quad \cos x = \frac{1}{\sec x},$ $\tan x = \frac{1}{\cot x}, \quad \sin^2 x + \cos^2 x = 1,$ $1 + \tan^2 x = \sec^2 x, \quad 1 + \cot^2 x = \csc^2 x,$ $\sin x = \cos\left(\frac{\pi}{2} - x\right), \quad \sin x = \sin(\pi - x),$ $\cos x = -\cos(\pi - x), \quad \tan x = \cot\left(\frac{\pi}{2} - x\right),$ $\cot x = -\cot(\pi - x), \quad \csc x = \cot \frac{x}{2} - \cot x,$ $\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$ $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$ $\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$ $\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$ $\sin 2x = 2 \sin x \cos x, \quad \sin 2x = \frac{2 \tan x}{1 + \tan^2 x},$ $\cos 2x = \cos^2 x - \sin^2 x, \quad \cos 2x = 2 \cos^2 x - 1,$ $\cos 2x = 1 - 2 \sin^2 x, \quad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$ $\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}, \quad \cot 2x = \frac{\cot^2 x - 1}{2 \cot x},$ $\sin(x+y) \sin(x-y) = \sin^2 x - \sin^2 y,$ $\cos(x+y) \cos(x-y) = \cos^2 x - \sin^2 y.$ <p>Euler's equation: $e^{ix} = \cos x + i \sin x, \quad e^{i\pi} = -1.$</p>	<p>Multiplication: $C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}.$</p> <p>Determinants: $\det A \neq 0$ iff A is non-singular. $\det A \cdot B = \det A \cdot \det B,$ $\det A = \sum_{\pi} \prod_{i=1}^n \text{sign}(\pi) a_{i,\pi(i)}.$</p> <p>$2 \times 2$ and 3×3 determinant: $\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$ $\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$ $= aei + bfg + cdh - ceg - fha - ibd.$</p> <p>Permanents: $\text{perm } A = \sum_{\pi} \prod_{i=1}^n a_{i,\pi(i)}.$</p>	<div></div> <p>Law of cosines: $c^2 = a^2 + b^2 - 2ab \cos C.$</p> <p>Area: $A = \frac{1}{2}hc,$ $= \frac{1}{2}ab \sin C,$ $= \frac{c^2 \sin A \sin B}{2 \sin C}.$</p> <p>Heron's formula: $A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$ $s = \frac{1}{2}(a + b + c),$ $s_a = s - a,$ $s_b = s - b,$ $s_c = s - c.$</p> <p>More identities: $\sin \frac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$ $\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$ $\tan \frac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$ $= \frac{1 - \cos x}{\sin x},$ $= \frac{\sin x}{1 + \cos x},$ $\cot \frac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$ $= \frac{1 + \cos x}{\sin x},$ $= \frac{\sin x}{1 - \cos x},$ $\sin x = \frac{e^{ix} - e^{-ix}}{2i},$ $\cos x = \frac{e^{ix} + e^{-ix}}{2},$ $\tan x = -i \frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$ $= -i \frac{e^{2ix} - 1}{e^{2ix} + 1},$ $\sin x = \frac{\sinh ix}{i},$ $\cos x = \cosh ix,$ $\tan x = \frac{\tanh ix}{i}.$</p>																									
<p>Hyperbolic Functions</p> <p>Definitions: $\sinh x = \frac{e^x - e^{-x}}{2}, \quad \cosh x = \frac{e^x + e^{-x}}{2},$ $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{csch } x = \frac{1}{\sinh x},$ $\text{sech } x = \frac{1}{\cosh x}, \quad \coth x = \frac{1}{\tanh x}.$</p> <p>Identities: $\cosh^2 x - \sinh^2 x = 1, \quad \tanh^2 x + \text{sech}^2 x = 1,$ $\coth^2 x - \text{csch}^2 x = 1, \quad \sinh(-x) = -\sinh x,$ $\cosh(-x) = \cosh x, \quad \tanh(-x) = -\tanh x,$ $\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$ $\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$ $\sinh 2x = 2 \sinh x \cosh x,$ $\cosh 2x = \cosh^2 x + \sinh^2 x,$ $\cosh x + \sinh x = e^x, \quad \cosh x - \sinh x = e^{-x},$ $(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$ $2 \sinh^2 \frac{x}{2} = \cosh x - 1, \quad 2 \cosh^2 \frac{x}{2} = \cosh x + 1.$</p>																											
<table><tr><td>θ</td><td>$\sin \theta$</td><td>$\cos \theta$</td><td>$\tan \theta$</td><td rowspan="6">... in mathematics you don't understand things, you just get used to them. – J. von Neumann</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>$\frac{\pi}{6}$</td><td>$\frac{1}{2}$</td><td>$\frac{\sqrt{3}}{2}$</td><td>$\frac{\sqrt{3}}{3}$</td></tr><tr><td>$\frac{\pi}{4}$</td><td>$\frac{\sqrt{2}}{2}$</td><td>$\frac{\sqrt{2}}{2}$</td><td>1</td></tr><tr><td>$\frac{\pi}{3}$</td><td>$\frac{\sqrt{3}}{2}$</td><td>$\frac{1}{2}$</td><td>$\sqrt{3}$</td></tr><tr><td>$\frac{\pi}{2}$</td><td>1</td><td>0</td><td>∞</td></tr></table>			θ	$\sin \theta$	$\cos \theta$	$\tan \theta$... in mathematics you don't understand things, you just get used to them. – J. von Neumann	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	∞
θ	$\sin \theta$	$\cos \theta$	$\tan \theta$... in mathematics you don't understand things, you just get used to them. – J. von Neumann																							
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	∞																								
v2.02 ©1994 by Steve Seiden sseiden@acm.org http://www.csc.lsu.edu/~seiden																											

Theoretical Computer Science Cheat Sheet										
Number Theory	Graph Theory									
<p>The Chinese remainder theorem: There exists a number C such that:</p> $C \equiv r_1 \bmod m_1$ \vdots $C \equiv r_n \bmod m_n$ <p>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</p> $\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$ <p>Euler's theorem: If a and b are relatively prime then</p> $1 \equiv a^{\phi(b)} \bmod b.$ <p>Fermat's theorem:</p> $1 \equiv a^{p-1} \bmod p.$ <p>The Euclidean algorithm: if $a > b$ are integers then</p> $\gcd(a, b) = \gcd(a \bmod b, b).$ <p>If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then</p> $S(x) = \sum_{d x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$ <p>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</p> $(n - 1)! \equiv -1 \bmod n.$ <p>Möbius inversion:</p> $\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$ <p>If</p> $G(a) = \sum_{d a} F(d),$ <p>then</p> $F(a) = \sum_{d a} \mu(d) G\left(\frac{a}{d}\right).$ <p>Prime numbers:</p> $p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$ $\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$	<p>Definitions:</p> <p><i>Loop</i> An edge connecting a vertex to itself.</p> <p><i>Directed</i> Each edge has a direction.</p> <p><i>Simple</i> Graph with no loops or multi-edges.</p> <p><i>Walk</i> A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$.</p> <p><i>Trail</i> A walk with distinct edges.</p> <p><i>Path</i> A trail with distinct vertices.</p> <p><i>Connected</i> A graph where there exists a path between any two vertices.</p> <p><i>Component</i> A maximal connected subgraph.</p> <p><i>Tree</i> A connected acyclic graph.</p> <p><i>Free tree</i> A tree with no root.</p> <p><i>DAG</i> Directed acyclic graph.</p> <p><i>Eulerian</i> Graph with a trail visiting each edge exactly once.</p> <p><i>Hamiltonian</i> Graph with a cycle visiting each vertex exactly once.</p> <p><i>Cut</i> A set of edges whose removal increases the number of components.</p> <p><i>Cut-set</i> A minimal cut.</p> <p><i>Cut edge</i> A size 1 cut.</p> <p><i>k-Connected</i> A graph connected with the removal of any $k - 1$ vertices.</p> <p><i>k-Tough</i> $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq S$.</p> <p><i>k-Regular</i> A graph where all vertices have degree k.</p> <p><i>k-Factor</i> A k-regular spanning subgraph.</p> <p><i>Matching</i> A set of edges, no two of which are adjacent.</p> <p><i>Clique</i> A set of vertices, all of which are adjacent.</p> <p><i>Ind. set</i> A set of vertices, none of which are adjacent.</p> <p><i>Vertex cover</i> A set of vertices which cover all edges.</p> <p><i>Planar graph</i> A graph which can be embedded in the plane.</p> <p><i>Plane graph</i> An embedding of a planar graph.</p> <hr/> $\sum_{v \in V} \deg(v) = 2m.$ <p>If G is planar then $n - m + f = 2$, so</p> $f \leq 2n - 4, \quad m \leq 3n - 6.$ <p>Any planar graph has a vertex with degree ≤ 5.</p>	<p>Notation:</p> <p>$E(G)$ Edge set</p> <p>$V(G)$ Vertex set</p> <p>$c(G)$ Number of components</p> <p>$G[S]$ Induced subgraph</p> <p>$\deg(v)$ Degree of v</p> <p>$\Delta(G)$ Maximum degree</p> <p>$\delta(G)$ Minimum degree</p> <p>$\chi(G)$ Chromatic number</p> <p>$\chi_E(G)$ Edge chromatic number</p> <p>G^c Complement graph</p> <p>K_n Complete graph</p> <p>K_{n_1, n_2} Complete bipartite graph</p> <p>$r(k, \ell)$ Ramsey number</p> <hr/> <p>Geometry</p> <p>Projective coordinates: triples (x, y, z), not all x, y and z zero.</p> $(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$ <table><tr><th>Cartesian</th><th>Projective</th></tr><tr><td>(x, y)</td><td>$(x, y, 1)$</td></tr><tr><td>$y = mx + b$</td><td>$(m, -1, b)$</td></tr><tr><td>$x = c$</td><td>$(1, 0, -c)$</td></tr></table> <p>Distance formula, L_p and L_∞ metric:</p> $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$ $[x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p},$ $\lim_{p \rightarrow \infty} [x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p}.$ <p>Area of triangle $(x_0, y_0), (x_1, y_1)$ and (x_2, y_2):</p> $\frac{1}{2} \text{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$ <p>Angle formed by three points:</p>  $\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$ <p>Line through two points (x_0, y_0) and (x_1, y_1):</p> $\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$ <p>Area of circle, volume of sphere:</p> $A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$ <hr/> <p>If I have seen farther than others, it is because I have stood on the shoulders of giants.</p> <p>– Issac Newton</p>	Cartesian	Projective	(x, y)	$(x, y, 1)$	$y = mx + b$	$(m, -1, b)$	$x = c$	$(1, 0, -c)$
Cartesian	Projective									
(x, y)	$(x, y, 1)$									
$y = mx + b$	$(m, -1, b)$									
$x = c$	$(1, 0, -c)$									

Theoretical Computer Science Cheat Sheet	
π	Calculus
<p>Wallis' identity:</p> $\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$ <p>Brouncker's continued fraction expansion:</p> $\frac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \cdots}}}}$ <p>Gregory's series:</p> $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$ <p>Newton's series:</p> $\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$ <p>Sharp's series:</p> $\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$ <p>Euler's series:</p> $\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$ $\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$ $\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$	<p>Derivatives:</p> <ol style="list-style-type: none"> $\frac{d(cu)}{dx} = c \frac{du}{dx}$, $\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$, $\frac{d(uv)}{dx} = u \frac{dv}{dx} + v \frac{du}{dx}$, $\frac{d(u^n)}{dx} = nu^{n-1} \frac{du}{dx}$, $\frac{d(u/v)}{dx} = \frac{v(\frac{du}{dx}) - u(\frac{dv}{dx})}{v^2}$, $\frac{d(e^{cu})}{dx} = ce^{cu} \frac{du}{dx}$, $\frac{d(\ln u)}{dx} = \frac{1}{u} \frac{du}{dx}$, $\frac{d(\sin u)}{dx} = \cos u \frac{du}{dx}$, $\frac{d(\cos u)}{dx} = -\sin u \frac{du}{dx}$, $\frac{d(\tan u)}{dx} = \sec^2 u \frac{du}{dx}$, $\frac{d(\cot u)}{dx} = -\csc^2 u \frac{du}{dx}$, $\frac{d(\sec u)}{dx} = \tan u \sec u \frac{du}{dx}$, $\frac{d(\csc u)}{dx} = -\cot u \csc u \frac{du}{dx}$, $\frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}} \frac{du}{dx}$, $\frac{d(\arccos u)}{dx} = \frac{-1}{\sqrt{1-u^2}} \frac{du}{dx}$, $\frac{d(\arctan u)}{dx} = \frac{1}{1+u^2} \frac{du}{dx}$, $\frac{d(\operatorname{arccot} u)}{dx} = \frac{-1}{1+u^2} \frac{du}{dx}$, $\frac{d(\operatorname{arcsec} u)}{dx} = \frac{1}{u\sqrt{1-u^2}} \frac{du}{dx}$, $\frac{d(\operatorname{arccsc} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx}$, $\frac{d(\sinh u)}{dx} = \cosh u \frac{du}{dx}$, $\frac{d(\cosh u)}{dx} = \sinh u \frac{du}{dx}$, $\frac{d(\tanh u)}{dx} = \operatorname{sech}^2 u \frac{du}{dx}$, $\frac{d(\coth u)}{dx} = -\operatorname{csch}^2 u \frac{du}{dx}$, $\frac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u \frac{du}{dx}$, $\frac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u \frac{du}{dx}$, $\frac{d(\operatorname{arcsinh} u)}{dx} = \frac{1}{\sqrt{1+u^2}} \frac{du}{dx}$, $\frac{d(\operatorname{arccosh} u)}{dx} = \frac{1}{\sqrt{u^2-1}} \frac{du}{dx}$, $\frac{d(\operatorname{arctanh} u)}{dx} = \frac{1}{1-u^2} \frac{du}{dx}$, $\frac{d(\operatorname{arccoth} u)}{dx} = \frac{1}{u^2-1} \frac{du}{dx}$, $\frac{d(\operatorname{arcsech} u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx}$, $\frac{d(\operatorname{arccsch} u)}{dx} = \frac{-1}{ u \sqrt{1+u^2}} \frac{du}{dx}$. <p>Integrals:</p> <ol style="list-style-type: none"> $\int cu \, dx = c \int u \, dx$, $\int (u+v) \, dx = \int u \, dx + \int v \, dx$, $\int x^n \, dx = \frac{1}{n+1} x^{n+1}$, $n \neq -1$, $\int \frac{1}{x} \, dx = \ln x$, $\int e^x \, dx = e^x$, $\int \frac{dx}{1+x^2} = \arctan x$, $\int u \frac{dv}{dx} \, dx = uv - \int v \frac{du}{dx} \, dx$, $\int \sin x \, dx = -\cos x$, $\int \cos x \, dx = \sin x$, $\int \tan x \, dx = -\ln \cos x$, $\int \cot x \, dx = \ln \cos x$, $\int \sec x \, dx = \ln \sec x + \tan x$, $\int \csc x \, dx = \ln \csc x + \cot x$, $\int \arcsin \frac{x}{a} \, dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}$, $a > 0$,
<p>Partial Fractions</p> <p>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</p> $\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$ <p>where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:</p> $\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$ <p>where</p> $A = \left[\frac{N(x)}{D(x)} \right]_{x=a}.$ <p>For a repeated factor:</p> $\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$ <p>where</p> $A_k = \frac{1}{k!} \left[\frac{d^k}{dx^k} \left(\frac{N(x)}{D(x)} \right) \right]_{x=a}.$	
<p>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.</p> <p>– George Bernard Shaw</p>	

Theoretical Computer Science Cheat Sheet

Calculus Cont.

15. $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$
16. $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$
17. $\int \sin^2(ax) dx = \frac{1}{2a} (ax - \sin(ax) \cos(ax)),$
18. $\int \cos^2(ax) dx = \frac{1}{2a} (ax + \sin(ax) \cos(ax)),$
19. $\int \sec^2 x dx = \tan x,$
20. $\int \csc^2 x dx = -\cot x,$
21. $\int \sin^n x dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx,$
22. $\int \cos^n x dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx,$
23. $\int \tan^n x dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x dx, \quad n \neq 1,$
24. $\int \cot^n x dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x dx, \quad n \neq 1,$
25. $\int \sec^n x dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x dx, \quad n \neq 1,$
26. $\int \csc^n x dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x dx, \quad n \neq 1,$
27. $\int \sinh x dx = \cosh x,$
28. $\int \cosh x dx = \sinh x,$
29. $\int \tanh x dx = \ln |\cosh x|,$
30. $\int \coth x dx = \ln |\sinh x|,$
31. $\int \operatorname{sech} x dx = \arctan \sinh x,$
32. $\int \operatorname{csch} x dx = \ln \left| \tanh \frac{x}{2} \right|,$
33. $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2} x,$
34. $\int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2} x,$
35. $\int \operatorname{sech}^2 x dx = \tanh x,$
36. $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$
37. $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$
38. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln \left(x + \sqrt{a^2 + x^2} \right), \quad a > 0,$
40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$
41. $\int \sqrt{a^2 - x^2} dx = \frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
42. $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8} (5a^2 - 2x^2) \sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
43. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$
44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|,$
45. $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 - x^2}},$
46. $\int \sqrt{a^2 \pm x^2} dx = \frac{x}{2} \sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left| x + \sqrt{a^2 \pm x^2} \right|,$
47. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left| x + \sqrt{x^2 - a^2} \right|, \quad a > 0,$
48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left| \frac{x}{a+bx} \right|,$
49. $\int x \sqrt{a+bx} dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$
50. $\int \frac{\sqrt{a+bx}}{x} dx = 2\sqrt{a+bx} + a \int \frac{1}{x\sqrt{a+bx}} dx,$
51. $\int \frac{x}{\sqrt{a+bx}} dx = \frac{1}{\sqrt{2}} \ln \left| \frac{\sqrt{a+bx} - \sqrt{a}}{\sqrt{a+bx} + \sqrt{a}} \right|, \quad a > 0,$
52. $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
53. $\int x \sqrt{a^2 - x^2} dx = -\frac{1}{3} (a^2 - x^2)^{3/2},$
54. $\int x^2 \sqrt{a^2 - x^2} dx = \frac{x}{8} (2x^2 - a^2) \sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
55. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
56. $\int \frac{x dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$
57. $\int \frac{x^2 dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
58. $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left| \frac{a + \sqrt{a^2 + x^2}}{x} \right|,$
59. $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$
60. $\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2},$
61. $\int \frac{dx}{x \sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

$$\begin{aligned}
62. \int \frac{dx}{x\sqrt{x^2-a^2}} &= \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0, & 63. \int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} &= \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x}, \\
64. \int \frac{x dx}{\sqrt{x^2 \pm a^2}} &= \sqrt{x^2 \pm a^2}, & 65. \int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx &= \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3}, \\
66. \int \frac{dx}{ax^2 + bx + c} &= \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left| \frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}} \right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases} \\
67. \int \frac{dx}{\sqrt{ax^2 + bx + c}} &= \begin{cases} \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c} \right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-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{4ac - 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} \frac{-1}{\sqrt{c}} \ln \left| \frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x} \right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases} \\
71. \int x^3 \sqrt{x^2 + a^2} dx &= (\frac{1}{3}x^2 - \frac{2}{15}a^2)(x^2 + a^2)^{3/2}, \\
72. \int x^n \sin(ax) dx &= -\frac{1}{a}x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) dx, \\
73. \int x^n \cos(ax) dx &= \frac{1}{a}x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) dx, \\
74. \int x^n e^{ax} dx &= \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx, \\
75. \int x^n \ln(ax) dx &= x^{n+1} \left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2} \right), \\
76. \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.
\end{aligned}$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$

$$\mathbb{E} f(x) = f(x+1).$$

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:

$$\Delta(cu) = c\Delta u, \quad \Delta(u+v) = \Delta u + \Delta v,$$

$$\Delta(uv) = u\Delta v + \mathbb{E} v \Delta u,$$

$$\Delta(x^n) = nx^{n-1},$$

$$\Delta(H_x) = x^{-1}, \quad \Delta(2^x) = 2^x,$$

$$\Delta(c^x) = (c-1)c^x, \quad \Delta\binom{x}{m} = \binom{x}{m-1}.$$

Sums:

$$\sum cu \delta x = c \sum u \delta x,$$

$$\sum(u+v) \delta x = \sum u \delta x + \sum v \delta x,$$

$$\sum u \Delta v \delta x = uv - \sum \mathbb{E} v \Delta u \delta x,$$

$$\sum x^n \delta x = \frac{x^{n+1}}{n+1}, \quad \sum x^{-1} \delta x = H_x,$$

$$\sum c^x \delta x = \frac{c^x}{c-1}, \quad \sum \binom{x}{m} \delta x = \binom{x}{m+1}.$$

Falling Factorial Powers:

$$x^{\underline{n}} = x(x-1) \cdots (x-n+1), \quad n > 0,$$

$$x^{\underline{0}} = 1,$$

$$x^{\underline{n}} = \frac{1}{(x+1) \cdots (x+|n|)}, \quad n < 0,$$

$$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$$

Rising Factorial Powers:

$$x^{\overline{n}} = x(x+1) \cdots (x+n-1), \quad n > 0,$$

$$x^{\overline{0}} = 1,$$

$$x^{\overline{n}} = \frac{1}{(x-1) \cdots (x-|n|)}, \quad n < 0,$$

$$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$$

Conversion:

$$x^{\underline{n}} = (-1)^n (-x)^{\overline{n}} = (x-n+1)^{\overline{n}}$$

$$= 1/(x+1)^{\overline{-n}},$$

$$x^{\overline{n}} = (-1)^n (-x)^{\underline{n}} = (x+n-1)^{\underline{n}}$$

$$= 1/(x-1)^{\underline{-n}},$$

$$x^n = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \left[\begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[\begin{matrix} n \\ k \end{matrix} \right] x^k.$$

$$\begin{aligned}
x^1 &= x^{\underline{1}} & x^{\overline{1}} \\
x^2 &= x^{\underline{2}} + x^{\underline{1}} & x^{\overline{2}} - x^{\overline{1}} \\
x^3 &= x^{\underline{3}} + 3x^{\underline{2}} + x^{\underline{1}} & x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}} \\
x^4 &= x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}} & x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}} \\
x^5 &= x^{\underline{5}} + 15x^{\underline{4}} + 25x^{\underline{3}} + 10x^{\underline{2}} + x^{\underline{1}} & x^{\overline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}} \\
x^{\overline{1}} &= x^1 & x^{\underline{1}} = x^1 \\
x^{\overline{2}} &= x^2 + x^1 & x^{\underline{2}} = x^2 - x^1 \\
x^{\overline{3}} &= x^3 + 3x^2 + 2x^1 & x^{\underline{3}} = x^3 - 3x^2 + 2x^1 \\
x^{\overline{4}} &= x^4 + 6x^3 + 11x^2 + 6x^1 & x^{\underline{4}} = x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\overline{5}} &= x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\underline{5}} = x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{aligned}$$

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 + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\ \ln(1+x) &= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}, \\ \ln \frac{1}{1-x} &= x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i}, \\ \sin x &= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\ \cos x &= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\ \tan^{-1} x &= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\ (1+x)^n &= 1 + nx + \frac{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 + \binom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\ \frac{x}{e^x - 1} &= 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{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 + \binom{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 + \frac{3}{2}x^2 + \frac{11}{6}x^3 + \frac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i, \\ \frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2 &= \frac{1}{2}x^2 + \frac{3}{4}x^3 + \frac{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:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$x A'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x) dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

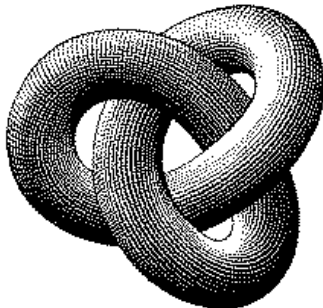
Summation: If $b_i = \sum_{j=0}^i a_j$ then

$$B(x) = \frac{1}{1-x} A(x).$$

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		Escher's Knot																																																																																																				
<div>Expansions:</div> <div>$\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,$$x^{\overline{n}} = \sum_{i=0}^{\infty} \left[\begin{matrix} n \\ i \end{matrix} \right] x^i,$$\left(\ln \frac{1}{1-x} \right)^n = \sum_{i=0}^{\infty} \left[\begin{matrix} i \\ n \end{matrix} \right] \frac{n! x^i}{i!},$$\tan x = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i} (2^{2i} - 1) B_{2i} x^{2i-1}}{(2i)!},$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\mu(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) = \sum_{d n} 1,$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \sum_{d n} d,$$\zeta(2n) = \frac{2^{2n-1} B_{2n} \pi^{2n}}{(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,$$\left(\frac{\arcsin x}{x} \right)^2 = \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.$</div>		<div></div>																																																																																																				
		Stieltjes Integration																																																																																																				
		<div>If G is continuous in the interval $[a, b]$ and F is nondecreasing then</div> <div>$\int_a^b G(x) dF(x)$</div> <div>exists. If $a \leq b \leq c$ then</div> <div>$\int_a^c G(x) dF(x) = \int_a^b G(x) dF(x) + \int_b^c G(x) dF(x).$</div> <div>If the integrals involved exist</div> <div>$\int_a^b (G(x) + H(x)) dF(x) = \int_a^b G(x) dF(x) + \int_a^b H(x) dF(x),$$\int_a^b G(x) d(F(x) + H(x)) = \int_a^b G(x) dF(x) + \int_a^b G(x) dH(x),$$\int_a^b c \cdot G(x) dF(x) = \int_a^b G(x) d(c \cdot F(x)) = c \int_a^b G(x) dF(x),$$\int_a^b G(x) dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x) dG(x).$</div> <div>If the integrals involved exist, and F possesses a derivative F' at every point in $[a, b]$ then</div> <div>$\int_a^b G(x) dF(x) = \int_a^b G(x) F'(x) dx.$</div>																																																																																																				
Cramer's Rule		Fibonacci Numbers																																																																																																				
<div>If we have equations:</div> <div>$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 \quad \quad \quad \vdots$$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$</div> <div>Let $A = (a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \neq 0$. Let A_i be A with column i replaced by B. Then</div> <div>$x_i = \frac{\det A_i}{\det A}.$</div>		<div><table><tr><td>00</td><td>47</td><td>18</td><td>76</td><td>29</td><td>93</td><td>85</td><td>34</td><td>61</td><td>52</td></tr><tr><td>86</td><td>11</td><td>57</td><td>28</td><td>70</td><td>39</td><td>94</td><td>45</td><td>02</td><td>63</td></tr><tr><td>95</td><td>80</td><td>22</td><td>67</td><td>38</td><td>71</td><td>49</td><td>56</td><td>13</td><td>04</td></tr><tr><td>59</td><td>96</td><td>81</td><td>33</td><td>07</td><td>48</td><td>72</td><td>60</td><td>24</td><td>15</td></tr><tr><td>73</td><td>69</td><td>90</td><td>82</td><td>44</td><td>17</td><td>58</td><td>01</td><td>35</td><td>26</td></tr><tr><td>68</td><td>74</td><td>09</td><td>91</td><td>83</td><td>55</td><td>27</td><td>12</td><td>46</td><td>30</td></tr><tr><td>37</td><td>08</td><td>75</td><td>19</td><td>92</td><td>84</td><td>66</td><td>23</td><td>50</td><td>41</td></tr><tr><td>14</td><td>25</td><td>36</td><td>40</td><td>51</td><td>62</td><td>03</td><td>77</td><td>88</td><td>99</td></tr><tr><td>21</td><td>32</td><td>43</td><td>54</td><td>65</td><td>06</td><td>10</td><td>89</td><td>97</td><td>78</td></tr><tr><td>42</td><td>53</td><td>64</td><td>05</td><td>16</td><td>20</td><td>31</td><td>98</td><td>79</td><td>87</td></tr></table></div> <div>1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...</div> <div>Definitions:</div> <div>$F_i = F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1,$$F_{-i} = (-1)^{i-1} F_i,$$F_i = \frac{1}{\sqrt{5}} \left(\phi^i - \hat{\phi}^i \right),$</div> <div>Cassini's identity: for $i > 0$:</div> <div>$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$</div> <div>Additive rule:</div> <div>$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$$F_{2n} = F_n F_{n+1} + F_{n-1} F_n.$</div> <div>Calculation by matrices:</div> <div>$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$</div>	00	47	18	76	29	93	85	34	61	52	86	11	57	28	70	39	94	45	02	63	95	80	22	67	38	71	49	56	13	04	59	96	81	33	07	48	72	60	24	15	73	69	90	82	44	17	58	01	35	26	68	74	09	91	83	55	27	12	46	30	37	08	75	19	92	84	66	23	50	41	14	25	36	40	51	62	03	77	88	99	21	32	43	54	65	06	10	89	97	78	42	53	64	05	16	20	31	98	79	87
00	47	18	76	29	93	85	34	61	52																																																																																													
86	11	57	28	70	39	94	45	02	63																																																																																													
95	80	22	67	38	71	49	56	13	04																																																																																													
59	96	81	33	07	48	72	60	24	15																																																																																													
73	69	90	82	44	17	58	01	35	26																																																																																													
68	74	09	91	83	55	27	12	46	30																																																																																													
37	08	75	19	92	84	66	23	50	41																																																																																													
14	25	36	40	51	62	03	77	88	99																																																																																													
21	32	43	54	65	06	10	89	97	78																																																																																													
42	53	64	05	16	20	31	98	79	87																																																																																													
Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius. – William Blake (The Marriage of Heaven and Hell)		<div>The Fibonacci number system:</div> <div>Every integer n has a unique representation</div> <div>$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$</div> <div>where $k_i \geq k_{i+1} + 2$ for all i, $1 \leq i < m$ and $k_m \geq 2$.</div>																																																																																																				

