

Arondight's Standard Code Library^{*}

Shanghai Jiao Tong University

Dated: October 11, 2017

^{*}<https://www.github.com/footoredo/Arondight>

1	计算几何	2	9.5	线性规划转对偶	22
1.1	凸包	3	9.6	32-bit/64-bit 随机素数	23
1.2	三角形的心	4	9.7	NTT 素数及其原根	23
1.3	半平面交	4	9.8	Java Hints	23
1.4	圆交面积及重心	5			
1.5	三维向量绕轴旋转	5			
1.6	三维凸包	6			
2	数论	6			
2.1	$O(m^2 \log n)$ 求线性递推数列第 n 项	6			
2.2	求逆元	7			
2.3	中国剩余定理	7			
2.4	魔法 CRT	7			
2.5	素性测试	8			
2.6	质因数分解	8			
2.7	线下整点	8			
2.8	原根相关	8			
3	代数	8			
3.1	快速傅里叶变换	8			
3.2	快速数论变换	9			
3.3	自适应辛普森积分	9			
3.4	单纯形	10			
4	字符串	10			
4.1	后缀数组	10			
4.2	后缀自动机	11			
4.3	EX 后缀自动机	11			
4.4	后缀树	11			
4.5	回文自动机	12			
5	数据结构	12			
5.1	KD-Tree	12			
5.2	Treap	13			
5.3	Link/cut Tree	14			
5.4	树状数组查询第 k 小元素	15			
6	图论	15			
6.1	基础	15			
6.2	KM	15			
6.3	点双连通分量	16			
6.4	边双连通分量	17			
6.5	最小树形图	17			
6.6	带花树	18			
6.7	Dominator Tree	18			
6.8	无向图最小割	19			
6.9	重口味费用流	20			
6.10	2-SAT	20			
7	其他	21			
7.1	Dancing Links	21			
7.2	蔡勒公式	22			
8	技巧	22			
8.1	真正的释放 STL 容器内存空间	22			
8.2	无敌的大整数相乘取模	22			
8.3	无敌的读入优化	22			
8.4	梅森旋转算法	22			
9	提示	22			
9.1	控制 cout 输出实数精度	22			
9.2	vimrc	22			
9.3	让 make 支持 c++11	22			
9.4	tuple 相关	22			

计算几何

```

1 k+ktint n+nfsignp(nDB nxp) p{
2     kreturn p(nx o> nepsp) o- p(nx o< o-nepsp);
3 p}
4 nDB n+nfsqrtp(nDB nxp) p{
5     kreturn nsignp(nxp) o> l+m+mi0 o? nsqrtp(nxp) o: l+m+mi0p;
6 p}
7
8 kstruct nPoint p{
9     nDB nxp, nyp;
10    nPoint n+nfrotatep(nDB nangp) kconst p{ c+c1// 逆时针旋转 ang 弧度
11        kreturn nPointp(ncosp(nangp) o* nx o- nsinp(nangp) o* nyp,
12            ncosp(nangp) o* ny o+ nsinp(nangp) o* nxp);
13    p}
14    nPoint n+nfturn90p() kconst p{ c+c1// 逆时针旋转 90 度
15        kreturn nPointp(o-nyp, nxp);
16    p}
17    nPoint n+nfunitp() kconst p{
18        kreturn o*kthis o/ nlenp();
19    p}
20 p};
21 nDB n+nfdotp(kconst nPointo& nap, kconst nPointo& nbp) p{
22     kreturn nap.nx o* nbp.nx o+ nap.ny o* nbp.nyp;
23 p}
24 nDB n+nfdetp(kconst nPointo& nap, kconst nPointo& nbp) p{
25     kreturn nap.nx o* nbp.ny o- nap.ny o* nbp.nxp;
26 p}
27 c+cp#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
28 c+cp#define cross0p(p1,p2,p3) sign(cross(p1,p2,p3))
29 k+ktbool n+nfisLLp(kconst nLineo& nl1p, kconst nLineo& nl2p, nPointo& npp) p{ c+c1// 直线
    ↳ 与直线交点
30    nDB ns1 o= ndetp(nl2p.nb o- nl2p.nap, nl1p.na o- nl2p.nap),
31        ns2 o= o-ndetp(nl2p.nb o- nl2p.nap, nl1p.nb o- nl2p.nap);
32    kif p(o!nsignp(ns1 o+ ns2p)) kreturn n+nbfalsep;
33    np o= p(nl1p.na o* ns2 o+ nl1p.nb o* ns1p) o/ p(ns1 o+ ns2p);
34    kreturn n+nbtruep;
35 p}
36 k+ktbool n+nfonSeggp(kconst nLineo& nlp, kconst nPointo& npp) p{ c+c1// 点在线段上
37     kreturn nsignp(ndetp(np o- nlp.nap, nlp.nb o- nlp.nap)) o== l+m+mi0 o&& nsignp(ndotp(np o-
    ↳ nlp.nap, np o- nlp.nbp)) o<= l+m+mi0p;
38 p}
39 nPoint n+nfprojectionp(kconst nLine o& nlp, kconst nPointo& npp) p{
40     kreturn nlp.na o+ p(nlp.nb o- nlp.nap) o* p(ndotp(np o- nlp.nap, nlp.nb o- nlp.nap) o/
    ↳ p(nlp.nb o- nlp.nap).nlen2p());
41 p}
42 nDB n+nfdisToLinep(kconst nLineo& nlp, kconst nPointo& npp) p{ c+c1// 点到 * 直线 * 距离
43     kreturn nfabsp(ndetp(np o- nlp.nap, nlp.nb o- nlp.nap) o/ p(nlp.nb o- nlp.nap).nlenp());
44 p}
45 nDB n+nfdisToSeggp(kconst nLineo& nlp, kconst nPointo& npp) p{ c+c1// 点到线段距离
46     kreturn nsignp(ndotp(np o- nlp.nap, nlp.nb o- nlp.nap)) o* nsignp(ndotp(np o- nlp.nbp,
    ↳ nlp.na o- nlp.nbp)) o== l+m+mi1 o? ndisToLinep(nlp, npp) o: nstdo::nminp((np o-
    ↳ nlp.nap).nlenp(), p(np o- nlp.nbp).nlenp());

```

```

47 p}
48 c+c1// 圆与直线交点
49 k+ktbool n+nfisCLp(nCircle nap, nLine nlp, nPointo& np1p, nPointo& np2p) p{
50     nDB nx o= ndotp(nlp.na o- nap.nop, nlp.nb o- nlp.nap),
51     ny o= p(nlp.nb o- nlp.nap).nlen2p(),
52     nd o= nx o* nx o- ny o* p((nlp.na o- nap.nop).nlen2p() o- nap.nr o* nap.nrp);
53     kif p(nsignp(ndp) o< l+m+mi0p) kreturn n+nbfalsep;
54     nPoint np o= nlp.na o- p((nlp.nb o- nlp.nap) o* p(nx o/ nyp)), ndelta o= p(nlp.nb o-
    ↳ nlp.nap) o* p(nmsqrtp(ndp) o/ nyp);
55     np1 o= np o+ ndeltap; np2 o= np o- ndeltap;
56     kreturn n+nbtruep;
57 p}
58 c+c1// 圆与圆的交面积
59 nDB n+nfareaCCp(kconst nCircleo& nc1p, kconst nCircleo& nc2p) p{
60     nDB nd o= p(nc1p.no o- nc2p.nop).nlenp();
61     kif p(nsignp(nd o- p(nc1p.nr o+ nc2p.nrp)) o>= l+m+mi0p) kreturn l+m+mi0p;
62     kif p(nsignp(nd o- nstdo::nabsp(nc1p.nr o- nc2p.nrp)) o<= l+m+mi0p) p{
63         nDB nr o= nstdo::nminp(nc1p.nrp, nc2p.nrp);
64         kreturn nr o* nr o* nPIp;
65     p}
66     nDB nx o= p(nd o* nd o+ nc1p.nr o* nc1p.nr o- nc2p.nr o* nc2p.nrp) o/ p(l+m+mi2 o* ndp),
67     nt1 o= nacosp(nx o/ nc1p.nrp), nt2 o= nacosp((nd o- nxp) o/ nc2p.nrp);
68     kreturn nc1p.nr o* nc1p.nr o* nt1 o+ nc2p.nr o* nc2p.nr o* nt2 o- nd o* nc1p.nr o*
    ↳ nsinp(nt1p);
69 p}
70 c+c1// 圆与圆交点
71 k+ktbool n+nfisCCp(nCircle nap, nCircle nbp, nPo& np1p, nPo& np2p) p{
72     nDB ns1 o= p(nap.no o- nbp.nop).nlenp();
73     kif p(nsignp(ns1 o- nap.nr o- nbp.nrp) o> l+m+mi0 o|| nsignp(ns1 o- nstdo::nabsp(nap.nr o-
    ↳ nbp.nrp)) o< l+m+mi0p) kreturn n+nbfalsep;
74     nDB ns2 o= p(nap.nr o* nap.nr o- nbp.nr o* nbp.nrp) o/ ns1p;
75     nDB naa o= p(ns1 o+ ns2p) o* l+m+mf0.5p, nbb o= p(ns1 o- ns2p) o* l+m+mf0.5p;
76     nP no o= p(nbp.no o- nap.nop) o* p(naa o/ p(naa o+ nbbp)) o+ nap.nop;
77     nP ndelta o= p(nbp.no o- nap.nop).nunitp().nturn90p() o* nmsqrtp(nap.nr o* nap.nr o- naa o*
    ↳ naap);
78     np1 o= no o+ ndeltap, np2 o= no o- ndeltap;
79     kreturn n+nbtruep;
80 p}
81 c+c1// 求点到圆的切点, 按关于点的顺时针方向返回两个点
82 k+ktbool n+nftanCPp(kconst nCircle o&ncp, kconst nPoint o&np0p, nPoint o&np1p, nPoint o&np2p)
    ↳ p{
83     k+ktdouble nx o= p(np0 o- ncp.nop).nlen2p(), nd o= nx o- ncp.nr o* ncp.nrp;
84     kif p(nd o< nepsp) kreturn n+nbfalsep; c+c1// 点在圆上认为没有切点
85     nPoint np o= p(np0 o- ncp.nop) o* p(ncp.nr o* ncp.nr o/ nxp);
86     nPoint ndelta o= p((np0 o- ncp.nop) o* p(o-ncp.nr o* nsqrtp(ndp) o/ nxp)).nturn90p();
87     np1 o= ncp.no o+ np o+ ndeltap;
88     np2 o= ncp.no o+ np o- ndeltap;
89     kreturn n+nbtruep;
90 p}
91 c+c1// 求圆到圆的外共切线, 按关于 c1.o 的顺时针方向返回两条线
92 nvectoro<nLineo> nextanCCp(kconst nCircle o&nc1p, kconst nCircle o&nc2p) p{
93     nvectoro<nLineo> nretp;
94     kif p(nsignp(nc1p.nr o- nc2p.nrp) o== l+m+mi0p) p{
95         nPoint ndir o= nc2p.no o- nc1p.nop;

```

```

96     ndir o= p(ndir o* p(nc1p.nr o/ ndirp.nlenp())).nturn90p();
97     nret.pnush_backp(nLinep(nc1p.no o+ ndirp, nc2p.no o+ ndirp));
98     nret.pnush_backp(nLinep(nc1p.no o- ndirp, nc2p.no o- ndirp));
99 p} kelse p{
100     nPoint np o= p(nc1p.no o* o-nc2p.nr o+ nc2p.no o* nc1p.nrp) o/ p(nc1p.nr o- nc2p.nrp);
101     nPoint np1p, np2p, nq1p, nq2p;
102     kif p(NTanCPp(nc1p, npp, np1p, np2p) o&& nTanCPp(nc2p, npp, nq1p, nq2p)) p{
103         kif p(nc1p.nr o< nc2p.nrp) nswapp(np1p, np2p), nswapp(nq1p, nq2p);
104         nret.pnush_backp(nLinep(np1p, nq1p));
105         nret.pnush_backp(nLinep(np2p, nq2p));
106     p}
107 p}
108 kreturn nret;
109 p}
110 c+c1// 求圆到圆的内共切线, 按关于 c1.o 的顺时针方向返回两条线
111 nstdo::nvectoro<nLineo> nntanCCp(kconst nCircle o&nc1p, kconst nCircle o&nc2p) p{
112     nstdo::nvectoro<nLineo> nret;
113     nPoint np o= p(nc1p.no o* nc2p.nr o+ nc2p.no o* nc1p.nrp) o/ p(nc1p.nr o+ nc2p.nrp);
114     nPoint np1p, np2p, nq1p, nq2p;
115     kif p(NTanCPp(nc1p, npp, np1p, np2p) o&& nTanCPp(nc2p, npp, nq1p, nq2p)) p{ c+c1// 两圆相
        ↳ 切认为没有切线
116         nret.pnush_backp(nLinep(np1p, nq1p));
117         nret.pnush_backp(nLinep(np2p, nq2p));
118     p}
119     kreturn nret;
120 p}
121 k+ktbool ncontainp(nvectoro<nPointo> npolygonp, nPoint npp) p{ c+c1// 判断点 p 是否被多边
        ↳ 形包含, 包括落在边界上
122     k+ktint nret o= l+m+mi0p, nn o= npolygonp.nsize();
123     kforp(k+ktint ni o= l+m+mi0p; ni o< nnp; o++ nip) p{
124         nPoint nu o= npolygonp[nip], nv o= npolygonp[(ni o+ l+m+mi1p) o% nnp];
125         kif p(nonSegp(nLinep(nup, nvp), npp)) kreturn n+nbtruep; c+c1// Here I guess.
126         kif p(nsignp(nup.ny o- nvp.nyp) o<= l+m+mi0p) nswapp(nup, nvp);
127         kif p(nsignp(npp.ny o- nup.nyp) o> l+m+mi0 o|| nsignp(npp.ny o- nvp.nyp) o<= l+m+mi0p)
            ↳ kcontinuep;
128         nret o+= nsignp(ndetp(npp, nvp, nup)) o> l+m+mi0p;
129     p}
130     kreturn nret o& l+m+mi1p;
131 p}
132 c+c1// 用半平面 (q1,q2) 的逆时针方向去切凸多边形
133 nstdo::nvectoro<nPointo> nconvexCutp(kconst nstdo::nvectoro<nPointo>&npsp, nPoint nq1p,
        ↳ nPoint nq2p) p{
134     nstdo::nvectoro<nPointo> nqsp; k+ktint nn o= npsp.nsize();
135     kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; o++nip) p{
136         nPoint np1 o= npsp[nip], np2 o= npsp[(ni o+ l+m+mi1p) o% nnp];
137         k+ktint nd1 o= ncrossOpp(nq1p, nq2p, np1p), nd2 o= ncrossOpp(nq1p, nq2p, np2p);
138         kif p(nd1 o>= l+m+mi0p) nqsp.npush_backp(np1p);
139         kif p(nd1 o* nd2 o< l+m+mi0p) nqsp.npush_backp(nisSSp(np1p, np2p, nq1p, nq2p));
140     p}
141     kreturn nqsp;
142 p}
143 c+c1// 求凸包
144 nstdo::nvectoro<nPointo> nconvexHullp(nstdo::nvectoro<nPointo> npsp) p{
145     k+ktint nn o= npsp.nsize(); kif p(nn o<= l+m+mi1p) kreturn npsp;

```

```

146     nstdo::nsortp(npsp.nbeginp(), npsp.nendp());
147     nstdo::nvectoro<nPointo> nqsp;
148     kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nqsp.npush_backp(npsp[ni o++p]))
149         kwhile p(nqsp.nsize() o> l+m+mi1 o&& nsignp(ndetp(nqsp[nqsp.nsize() o- l+m+mi2p],
            ↳ nqsp.nbackp(), npsp[nip])) o<= l+m+mi0p)
150             nqsp.npop_backp();
151     kfor p(k+ktint ni o= nn o- l+m+mi2p, nt o= nqsp.nsize(); ni o>= l+m+mi0p;
            ↳ nqsp.npush_backp(npsp[ni o--p]))
152         kwhile p((k+ktintp)nqsp.nsize() o> nt o&& nsignp(ndetp(nqsp[nqsp.nsize() o- l+m+mi2p],
            ↳ nqsp.nbackp(), npsp[nip])) o<= l+m+mi0p)
153             nqsp.npop_backp();
154     kreturn nqsp;

```

凸包

```

1 c+c1// 凸包中的点按逆时针方向
2 kstruct nConvex p{
3     k+ktint nnp;
4     nstdo::nvectoro<nPointo> nap, nupperp, nlowerp;
5     k+ktvoid n+nfmakeshellp(kconst nstdo::nvectoro<nPointo>&npp,
6         nstdo::nvectoro<nPointo>&nshellp) p{ c+c1// p needs to be sorted.
7         nclearp(nshellp); k+ktint nn o= npp.nsize();
8         kfor p(k+ktint ni o= l+m+mi0p, nj o= l+m+mi0p; ni o< nnp; nio++p, njo++p) p{
9             kfor p(; nj o>= l+m+mi2 o&& nsignp(ndetp(nshellp[njo-l+m+mi1p] o-
            ↳ nshellp[njo-l+m+mi2p],
10                 npp[nip] o- nshellp[njo-l+m+mi2p])) o<= l+m+mi0p; o--njp) nshellp.npop_backp();
11                 nshellp.npush_backp(npp[nip]);
12     p}
13 p}
14 k+ktvoid n+nfmakesconvexp() p{
15     nstdo::nsortp(nap.nbeginp(), nap.nendp());
16     nmake_shellp(nap, nlowerp);
17     nstdo::nreversep(nap.nbeginp(), nap.nendp());
18     nmake_shellp(nap, nupperp);
19     na o= nlowerp; nap.npop_backp();
20     nap.ninsertp(nap.nendp(), nupperp.nbeginp(), nupperp.nendp());
21     kif p((k+ktintp)nap.nsize() o>= l+m+mi2p) nap.npop_backp();
22     nn o= nap.nsize();
23 p}
24 k+ktvoid n+nfinityp(kconst nstdo::nvectoro<nPointo>&n_ap) p{
25     nclearp(nap); na o= n_ap; nn o= nap.nsize();
26     nmake_convexp();
27 p}
28 k+ktvoid n+nfreaddp(k+ktint n_np) p{ c+c1// Won't make convex.
29     nclearp(nap); nn o= n_np; nap.nresize(nnp);
30     kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p)
31         nap[nip].nreadp();
32 p}
33 nstdo::npairo<nDBp, k+ktinto> nget_tangentp(
34     kconst nstdo::nvectoro<nPointo>&nconvexp, kconst nPointo&nvecp) p{
35     k+ktint nl o= l+m+mi0p, nr o= p(k+ktintp)nconvexp.nsize() o- l+m+mi2p;
36     nassertp(nr o>= l+m+mi0p);
37     kfor p(; nl o+ l+m+mi1 o< nr; p) p{
38         k+ktint nmid o= p(nl o+ nr) o/ l+m+mi2p;
39         kif p(nsignp(ndetp(nconvexp[nmid o+ l+m+mi1p] o- nconvexp[nmidp], nvecp)) o> l+m+mi0p)

```

```

40     nr o= nmidp;
41     kelse nl o= nmidp;
42 }
43 kreturn nstdo::nmaxp(nstdo::nmake_pairp(ndetp(nvecp, nconvexp[nrp]), nrp),
44     nstdo::nmake_pairp(ndetp(nvecp, nconvexp[l+m+mi0p]), l+m+mi0p));
45 p}
46 k+ktint nbinary_searchp(nPoint nup, nPoint nvp, k+ktint nlp, k+ktint nrp) p{
47     k+ktint ns1 o= nsignp(ndetp(nv o- nup, nap[nl o% nnp] o- nup));
48     kfor p(; nl o+ l+m+mi1 o< nrp; p) p{
49         k+ktint nmid o= p(nl o+ nrp) o/ l+m+mi2p;
50         k+ktint nsmid o= nsignp(ndetp(nv o- nup, nap[nmid o% nnp] o- nup));
51         kif p(nsmid o== ns1p) nl o= nmidp;
52         kelse nr o= nmidp;
53     }
54     kreturn nl o% nnp;
55 p}
56 c+c1// 求凸包上和向量 vec 叉积最大的点, 返回编号, 共线的多个切点返回任意一个
57 k+ktint nget_tangentp(nPoint nvecp) p{
58     nstdo::npairo<nDBp, k+ktinto> nret o= nget_tangentp(nupperp, nvecp);
59     nret.p.nsecond o= p(nret.p.nsecond o+ p(k+ktintp)nlowerp.nsizep()) o- l+m+mi1p) o% nnp;
60     nret o= nstdo::nmaxp(nret.p, nget_tangentp(nlowerp, nvecp));
61     kreturn nret.p.nsecondp;
62 p}
63 c+c1// 求凸包和直线 u, v 的交点, 如果不相交返回 false, 如果有则是和 (i,
    ↪ next(i)) 的交点, 交在点上不确定返回前后两条边其中之一
64 k+ktbool nget_intersectionp(nPoint nup, nPoint nvp, k+ktint o&ni0p, k+ktint o&ni1p) p{
65     k+ktint np0 o= nget_tangentp(nu o- nvp), np1 o= nget_tangentp(nv o- nup);
66     kif p(nsignp(ndetp(nv o- nup, nap[np0p] o- nup)) o* nsignp(ndetp(nv o- nup, nap[np1p] o-
    ↪ nup)) o<= l+m+mi0p) p{
67         kif p(np0 o> np1p) nstdo::nswapp(np0p, np1p);
68         ni0 o= nbinary_searchp(nup, nvp, np0p, np1p);
69         ni1 o= nbinary_searchp(nup, nvp, np1p, np0 o+ nnp);
70         kreturn n+nbttruep;
71     }
72     kelse kreturn n+nbfalsep;
73 p}
74 p};

```

三角形的心

```

1 nPoint n+nfinCenterp(kconst nPoint o&nAp, kconst nPoint o&nBp, kconst nPoint o&nCp) p{ c+c1//
    ↪ 内心
2     k+ktdouble na o= p(nB o- nCp).nlenp(), nb o= p(nC o- nAp).nlenp(), nc o= p(nA o-
    ↪ nBp).nlenp(),
3     ns o= nfabsp(ndetp(nB o- nAp, nC o- nAp)),
4     nr o= ns o/ npp;
5     kreturn p(nA o* na o+ nB o* nb o+ nC o* nc o) o/ p(na o+ nb o+ nc o);
6 p}
7 nPoint n+nfcircumCenterp(kconst nPoint o&nap, kconst nPoint o&nbp, kconst nPoint o&ncp) p{
    ↪ c+c1// 外心
8     nPoint nbb o= nb o- nap, ncc o= nc o- nap;
9     k+ktdouble ndb o= nbbp.nlen2p(), ndc o= nccp.nlen2p(), nd o= l+m+mi2 o* ndetp(nbbp, nccp);
10    kreturn na o- nPointp(nbbp.ny o* ndc o- nccp.ny o* ndbp, nccp.nx o* ndb o- nbbp.nx o* ndcp)
    ↪ o/ ndp;
11 p}

```

```

12 nPoint n+nfothroCenterp(kconst nPoint o&nap, kconst nPoint o&nbp, kconst nPoint o&ncp) p{
    ↪ c+c1// 垂心
13     nPoint nba o= nb o- nap, nca o= nc o- nap, nbc o= nb o- ncp;
14     k+ktdouble nY o= nbap.ny o* ncap.ny o* nbcp.nyp,
15     nA o= ncap.nx o* nbap.ny o- nbap.nx o* ncap.nyp,
16     nx0 o= p(nY o+ ncap.nx o* nbap.ny o* nbp.nx o- nbap.nx o* ncap.ny o* ncp.nxp) o/ nAp,
17     ny0 o= o-nbap.nx o* p(nx0 o- ncp.nxp) o/ nbap.ny o+ ncap.nyp;
18     kreturn nPointp(nx0p, ny0p);
19 p}

```

半平面交

```

1 kstruct nPoint p{
2     k+ktint nquadp() kconst p{ kreturn nsignp(nyp) o== l+m+mi1 o|| p(nsignp(nyp) o== l+m+mi0
    ↪ o&& nsignp(nxp) o>= l+m+mi0p);}
3 p};
4 kstruct nLine p{
5     k+ktbool nincluep(kconst nPoint o&npp) kconst p{ kreturn nsignp(ndetp(nb o- nap, np o-
    ↪ nap)) o> l+m+mi0p; p}
6     nline npushp() kconst{ c+c1// 将半平面向外推 eps
7         kconst k+ktdouble neps o= l+m+mf1e-6p;
8         nPoint ndelta o= p(nb o- nap).nturn90p().nnormp() o* neps;
9         kreturn n+nflinep(na o- ndeltap, nb o- ndeltap);
10    }
11 p};
12 k+ktbool n+nfsameDirp(kconst nLine o&nlp0, kconst nLine o&nlp1) p{ kreturn nparallelp(nl0p,
    ↪ nl1p) o&& nsignp(ndotp(nl0p.nb o- nl0p.nap, nl1p.nb o- nl1p.nap)) o== l+m+mi1p; p}
13 k+ktbool koperator o< p(kconst nPoint o&nap, kconst nPoint o&nbp) p{
14     kif p(nap.nquadp() o!= nbp.nquadp()) p{
15         kreturn nap.nquadp() o< nbp.nquadp();
16     } kelse p{
17         kreturn nsignp(ndetp(nap, nbp)) o> l+m+mi0p;
18     }
19 p}
20 k+ktbool koperator o< p(kconst nLine o&nlp0, kconst nLine o&nlp1) p{
21     kif p(nsameDirp(nl0p, nl1p)) p{
22         kreturn nl1p.nincluep(nl0p.nap);
23     } kelse p{
24         kreturn p(nl0p.nb o- nl0p.nap) o< p(nl1p.nb o- nl1p.nap);
25     }
26 p}
27 k+ktbool ncheckp(kconst nLine o&nup, kconst nLine o&nvp, kconst nLine o&nwp) p{ kreturn
    ↪ nwp.nincluep(nintersectp(nup, nvp)); p}
28 nvectoro<nPointo> nintersectionp(nvectoro<nLineo> o&nlp) p{
29     nsortp(nlp.nbeginp(), nlp.nendp());
30     ndequeo<nLineo> nqp;
31     kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nlp.nsizep(); o++nlp) p{
32         kif p(ni o&& nsameDirp(nlp[nip], nlp[ni o- l+m+mi1p])) p{
33             kcontinuep;
34         }
35         kwhile p(nqp.nsizep() o> l+m+mi1 o&& o!ncheckp(nqp[nqp.nsizep() o- l+m+mi2p],
    ↪ nqp[nqp.nsizep() o- l+m+mi1p], nlp[nip])) nqp.npop_backp();
36         kwhile p(nqp.nsizep() o> l+m+mi1 o&& o!ncheckp(nqp[l+m+mi1p], nqp[l+m+mi0p], nlp[nip]))
    ↪ nqp.npop_frontp();
37         nqp.npush_backp(nlp[nip]);

```

```

38 p}
39 kwhile p(nqp.nsize() o> l+m+mi2 o&& o!ncheckp(nqp[nqp.nsize() o- l+m+mi2p],
    ↳ nqp[nqp.nsize() o- l+m+mi1p], nqp[l+m+mi0p])) nqp.npop_backp();
40 kwhile p(nqp.nsize() o> l+m+mi2 o&& o!ncheckp(nqp[l+m+mi1p], nqp[l+m+mi0p],
    ↳ nqp[nqp.nsize() o- l+m+mi1p])) nqp.npop_frontp();
41 nvectoro<nPointo> nretp;
42 kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nqp.nsize(); o++njp)
    ↳ nretp.npush_backp(nintersectp(nqp[nip], nqp[(ni o+ l+m+mi1p) o% nqp.nsize()]));
43 kreturn nretp;
44 p}

```

圆交面积及重心

```

1 kstruct nEvent p{
2     nPoint npp;
3     k+ktdouble nangp;
4     k+ktint ndeltap;
5     nEvent p(nPoint np o= nPointp(l+m+mi0p, l+m+mi0p), k+ktdouble nang o= l+m+mi0p, k+ktdouble
    ↳ ndelta o= l+m+mi0p) o: npp(npp), nangp(nangp), ndeltap(ndeltap) p{
6 p};
7 k+ktbool koperator o< p(kconst nEvent o&nap, kconst nEvent o&nbp) p{
8     kreturn nap.nang o< nbp.nangp;
9 p}
10 k+ktvoid naddEventp(kconst nCircle o&nap, kconst nCircle o&nbp, nvectoro<nEvento> o&nevtp,
    ↳ k+ktint o&ncntp) p{
11     k+ktdouble nd2 o= p(nap.no o- nbp.nop).nlen2p(),
12     ndRatio o= p((nap.nr o- nbp.nrp) o* p(nap.nr o+ nbp.nrp) o/ nd2 o+ l+m+mi1p) o/
    ↳ l+m+mi2p,
13     npRatio o= nsqrtp(o-p(nd2 o- nsqrp(nap.nr o- nbp.nrp)) o* p(nd2 o- nsqrp(nap.nr o+
    ↳ nbp.nrp)) o/ p(nd2 o* nd2 o* l+m+mi4p));
14     nPoint nd o= nbp.no o- nap.nop, np o= ndp.nrotatep(nPI o/ l+m+mi2p),
15     nq0 o= nap.no o+ nd o* ndRatio o+ np o* npRatio,
16     nq1 o= nap.no o+ nd o* ndRatio o- np o* npRatio;
17     k+ktdouble nang0 o= p(nq0 o- nap.nop).nangp(),
18     nang1 o= p(nq1 o- nap.nop).nangp();
19     nevtp.npush_backp(nEventp(nq1p, nang1p, l+m+mi1p));
20     nevtp.npush_backp(nEventp(nq0p, nang0p, o-l+m+mi1p));
21     ncnt o+= nang1 o> nang0p;
22 p}
23 k+ktbool nissamep(kconst nCircle o&nap, kconst nCircle o&nbp) p{ kreturn nsignp((nap.no o-
    ↳ nbp.nop).nlenp()) o== l+m+mi0 o&& nsignp(nap.nr o- nbp.nrp) o== l+m+mi0p; p}
24 k+ktbool noverlapp(kconst nCircle o&nap, kconst nCircle o&nbp) p{ kreturn nsignp(nap.nr o-
    ↳ nbp.nr o- p(nap.no o- nbp.nop).nlenp()) o>= l+m+mi0p; p}
25 k+ktbool nintersectp(kconst nCircle o&nap, kconst nCircle o&nbp) p{ kreturn nsignp((nap.no o-
    ↳ nbp.nop).nlenp() o- nap.nr o- nbp.nrp) o< l+m+mi0p; p}
26 nCircle ncp[nNp];
27 k+ktdouble nareap[nNp]; c+c1// area[k] -> area of intersections >= k.
28 nPoint ncentroidp[nNp];
29 k+ktbool nkeep[nNp];
30 k+ktvoid n+nfaddp(k+ktint ncntp, nDB nap, nPoint ncp) p{
31     nareap[ncntp] o+= nap;
32     ncentroidp[ncntp] o= ncentroidp[ncntp] o+ nc o* nap;
33 p}
34 k+ktvoid n+nfsolvep(k+ktint nCp) p{
35     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nCp; o++ nip) p{
36         nareap[nip] o= l+m+mi0p;

```

```

37     ncentroidp[nip] o= nPointp(l+m+mi0p, l+m+mi0p);
38 p}
39 kfor p(k+ktint ni o= l+m+mi0p; ni o< nCp; o++njp) p{
40     k+ktint ncnt o= l+m+mi1p;
41     nvectoro<nEvento> nevtp;
42     kfor p(k+ktint nj o= l+m+mi0p; nj o< njp; o++njp) kif p(nissamep(ncp[nip], ncp[njp]))
    ↳ o++ncntp;
43     kfor p(k+ktint nj o= l+m+mi0p; nj o< nCp; o++njp) p{
44         kif p(nj o!= ni o&& o!nissamep(ncp[nip], ncp[njp]) o&& noverlapp(ncp[njp], ncp[nip]))
    ↳ p{
45             o++ncntp;
46         p}
47 p}
48 kfor p(k+ktint nj o= l+m+mi0p; nj o< nCp; o++njp) p{
49     kif p(nj o!= ni o&& o!noverlapp(ncp[njp], ncp[nip]) o&& o!noverlapp(ncp[nip], ncp[njp])
    ↳ o&& nintersectp(ncp[nip], ncp[njp])) p{
50         naddEventp(ncp[nip], ncp[njp], nevtp, ncntp);
51     p}
52 p}
53 kif p(nevtp.nsize() o== l+m+mi0up) p{
54     naddp(ncntp, nPI o* ncp[nip].nr o* ncp[nip].nrp, ncp[nip].nop);
55 p} kelse p{
56     nsortp(nevtp.nbeginp(), nevtp.nendp());
57     nevtp.npush_backp(nevtp.nfrontp());
58     kfor p(k+ktint nj o= l+m+mi0p; nj o+ l+m+mi1 o< p(k+ktintp)nevtp.nsize(); o++njp) p{
59         ncnt o+= nevtp[njp].ndeltap;
60         naddp(ncntp, ndetp(nevtp[njp].npp, nevtp[nj o+ l+m+mi1p].npp) o/ l+m+mi2p,
    ↳ p(nevtp[njp].np o+ nevtp[nj o+ l+m+mi1p].npp) o/ l+m+mi3p);
61         k+ktdouble nang o= nevtp[nj o+ l+m+mi1p].nang o- nevtp[njp].nangp;
62         kif p(nang o< l+m+mi0p) p{
63             nang o+= nPI o* l+m+mi2p;
64         p}
65         kif p(nsignp(nangp) o== l+m+mi0p) kcontinuep;
66         naddp(ncntp, nang o* ncp[nip].nr o* ncp[nip].nrp o/ l+m+mi2p, ncp[nip].no o+
            nPointp(nsinp(nang1p) o- nsinp(nang0p), o-ncosp(nang1p) o+ ncosp(nang0p))
    ↳ o* p(l+m+mi2 o/ p(l+m+mi3 o* nangp) o* ncp[nip].nrp));
67         naddp(ncntp, o-nsinp(nangp) o* ncp[nip].nr o* ncp[nip].nrp o/ l+m+mi2p, p(ncp[nip].no
    ↳ o+ nevtp[njp].np o+ nevtp[nj o+ l+m+mi1p].npp) o/ l+m+mi3p);
68     p}
69 p}
70 p}
71 p}
72 kfor p(k+ktint ni o= l+m+mi1p; ni o<= nCp; o++ nip)
73     kif p(nsignp(nareap[nip])) p{
74         ncentroidp[nip] o= ncentroidp[nip] o/ nareap[nip];
75     p}
76 p}

```

三维向量绕轴旋转

```

1 c+c1// 三维绕轴旋转，大拇指指向 axis 向量方向，四指弯曲方向转 w 弧度
2 nPoint n+nfrotatep(kconst nPointo& nsp, kconst nPointo& naxisp, nDB nwp) p{
3     nDB nx o= naxisp.nxp, ny o= naxisp.nyp, nz o= naxisp.nzp;
4     nDB ns1 o= nx o* nx o+ ny o* ny o+ nz o* nzp, nss1 o= nmsqrtp(ns1p),
5     ncosp o= ncosp(nwp), nsinw o= nsinp(nwp);
6     nDB nap[l+m+mi4p][l+m+mi4p];

```

```

7  nmemsetp(nap, l+m+mi0p, sizeof nap);
8  nap[l+m+mi3p][l+m+mi3p] o= l+m+mi1p;
9  nap[l+m+mi0p][l+m+mi0p] o= p((ny o* ny o+ nz o* nzp) o* ncosw o+ nx o* nxp) o/ ns1p;
10 nap[l+m+mi0p][l+m+mi1p] o= nx o* ny o* p(l+m+mi1 o- ncoswp) o/ ns1 o+ nz o* nsinw o/ nss1p;
11 nap[l+m+mi0p][l+m+mi2p] o= nx o* nz o* p(l+m+mi1 o- ncoswp) o/ ns1 o- ny o* nsinw o/ nss1p;
12 nap[l+m+mi1p][l+m+mi0p] o= nx o* ny o* p(l+m+mi1 o- ncoswp) o/ ns1 o- nz o* nsinw o/ nss1p;
13 nap[l+m+mi1p][l+m+mi1p] o= p((nx o* nx o+ nz o* nzp) o* ncosw o+ ny o* nyp) o/ ns1p;
14 nap[l+m+mi1p][l+m+mi2p] o= ny o* nz o* p(l+m+mi1 o- ncoswp) o/ ns1 o+ nx o* nsinw o/ nss1p;
15 nap[l+m+mi2p][l+m+mi0p] o= nx o* nz o* p(l+m+mi1 o- ncoswp) o/ ns1 o+ ny o* nsinw o/ nss1p;
16 nap[l+m+mi2p][l+m+mi1p] o= ny o* nz o* p(l+m+mi1 o- ncoswp) o/ ns1 o- nx o* nsinw o/ nss1p;
17 nap[l+m+mi2p][l+m+mi2p] o= p((nx o* nx o+ ny o* nyp) o* ncosp(nwp) o+ nz o* nzp) o/ ns1p;
18 nDB nansp[l+m+mi4p] o= p{l+m+mi0p, l+m+mi0p, l+m+mi0p, l+m+mi0p}, ncp[l+m+mi4p] o=
    ↪ p{nsp.nxp, nsp.nyp, nsp.nzp, l+m+mi1p};
19 kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi4p; o++ nip)
20     kfor p(k+ktint nj o= l+m+mi0p; nj o< l+m+mi4p; o++ njp)
21         nansp[nip] o+= nap[njp][nip] o* ncp[njp];
22 kreturn nPointp(nansp[l+m+mi0p], nansp[l+m+mi1p], nansp[l+m+mi2p]);
23 p}

```

三维凸包

```

1 k+kr__inline nP n+nfcrossp(kconst nPo& nap, kconst nPo& nbp) p{
2     kreturn nPp(
3         nap.ny o* nbp.nz o- nap.nz o* nbp.nyp,
4         nap.nz o* nbp.nx o- nap.nx o* nbp.nzp,
5         nap.nx o* nbp.ny o- nap.ny o* nbp.nx
6         p);
7 p}
8
9 k+kr__inline nDB n+nfmixp(kconst nPo& nap, kconst nPo& nbp, kconst nPo& ncp) p{
10     kreturn ndotp(ncrossp(nap, nbp), ncp);
11 p}
12
13 k+kr__inline nDB n+nfvolumep(kconst nPo& nap, kconst nPo& nbp, kconst nPo& ncp, kconst nPo&
    ↪ ndp) p{
14     kreturn nmixp(nb o- nap, nc o- nap, nd o- nap);
15 p}
16
17 kstruct nFace p{
18     k+ktint nap, nbp, ncp;
19     k+kr__inline n+nffacep() p{
20     k+kr__inline n+nffacep(k+ktint n_ap, k+ktint n_bp, k+ktint n_cp)o:
21         nap(n_ap), nbp(n_bp), ncp(n_cp) p{
22     k+kr__inline nDB n+nfareap() kconst p{
23         kreturn l+m+mf0.5 o* ncrossp(npp[nbp] o- npp[nap], npp[ncp] o- npp[nap]).nlenn();
24     p}
25     k+kr__inline nP n+nfnormalp() kconst p{
26         kreturn ncrossp(npp[nbp] o- npp[nap], npp[ncp] o- npp[nap]).nunitp();
27     p}
28     k+kr__inline nDB n+nfdisp(kconst nPo& np0p) kconst p{
29         kreturn ndotp(nnormalp(), np0 o- npp[nap]);
30     p}
31 p};
32
33 nstdo::nvectoro<nFaceo> nfacep, ntmpp; c+c1// Should be 0(n).
34 k+ktint nmarkp[nNp][nNp], nTimep, nnp;

```

```

35
36 k+kr__inline k+ktvoid n+nfaddp(k+ktint nvp) p{
37     o++ nTimep;
38     nclearp(ntmpp);
39     kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nfacep.nsizep(); o++ nip) p{
40         k+ktint na o= nfacep[nip].nap, nb o= nfacep[nip].nbp, nc o= nfacep[nip].ncp;
41         kif p(nsignp(nvolumep(npp[nvp], npp[nap], npp[nbp], npp[ncp])) o> l+m+mi0p) p{
42             nmarkp[nap][nbp] o= nmarkp[nbp][nap] o= nmarkp[nap][ncp] o=
43                 nmarkp[ncp][nap] o= nmarkp[nbp][ncp] o= nmarkp[ncp][nbp] o= nTimep;
44         p}
45         kelse p{
46             ntmpp.npush_backp(nfacep[nip]);
47         p}
48     p}
49     nclearp(nfacep); nface o= ntmpp;
50     kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)ntmpp.nsizep(); o++ nip) p{
51         k+ktint na o= nfacep[nip].nap, nb o= nfacep[nip].nbp, nc o= nfacep[nip].ncp;
52         kif p(nmarkp[nap][nbp] o== nTimep) nfacep.nemplace_backp(nvp, nbp, nap);
53         kif p(nmarkp[nbp][ncp] o== nTimep) nfacep.nemplace_backp(nvp, ncp, nbp);
54         kif p(nmarkp[ncp][nap] o== nTimep) nfacep.nemplace_backp(nvp, nap, ncp);
55         nassertp(nfacep.nsizep() o< l+m+mi500up);
56     p}
57 p}
58
59 k+ktvoid n+nfreorderp() p{
60     kfor p(k+ktint ni o= l+m+mi2p; ni o< nnp; o++ nip) p{
61         nP ntmp o= ncrossp(npp[nip] o- npp[l+m+mi0p], npp[nip] o- npp[l+m+mi1p]);
62         kif p(nsignp(ntmpp.nlenn())) p{
63             nstdo::nswapp(npp[nip], npp[l+m+mi2p]);
64             kfor p(k+ktint nj o= l+m+mi3p; nj o< nnp; o++ njp)
65                 kif p(nsignp(nvolumep(npp[l+m+mi0p], npp[l+m+mi1p], npp[l+m+mi2p], npp[njp])) p{
66                     nstdo::nswapp(npp[njp], npp[l+m+mi3p]);
67                     kreturnp;
68                 p}
69         p}
70     p}
71 p}
72
73 k+ktvoid n+nfbuild_convexp() p{
74     nreorderp();
75     nclearp(nfacep);
76     nfacep.nemplace_backp(l+m+mi0p, l+m+mi1p, l+m+mi2p);
77     nfacep.nemplace_backp(l+m+mi0p, l+m+mi2p, l+m+mi1p);
78     kfor p(k+ktint ni o= l+m+mi3p; ni o< nnp; o++ nip)
79         naddp(nip);
80 p}

```

数论

$O(m^2 \log n)$ 求线性递推数列第 n 项

Given a_0, a_1, \dots, a_{m-1}
 $a_n = c_0 \times a_{n-m} + \dots + c_{m-1} \times a_{n-1}$
 Solve for $a_n = v_0 \times a_0 + v_1 \times a_1 + \dots + v_{m-1} \times a_{m-1}$

```

1 k+ktvoid n+nflinear_recurrencep(k+ktlong k+ktlong nnp, k+ktint nmp, k+ktint nap[], k+ktint
  ↳ ncp[], k+ktint npp) p{
2   k+ktlong k+ktlong nvp[nMp] o= p{l+m+mi1 o% npp}, nup[nM o<< l+m+mi1p], nmsk o= o!!nnp;
3   kforp(k+ktlong k+ktlong nip(nnp); ni o> l+m+mi1p; ni o>= l+m+mi1p) p{
4     nmsk o<= l+m+mi1p;
5   p}
6   kforp(k+ktlong k+ktlong nxp(l+m+mi0p); nmskp; nmsk o>= l+m+mi1p, nx o<= l+m+mi1p) p{
7     nfill_n(p(nup, nm o<< l+m+mi1p, l+m+mi0p);
8     k+ktint nbp(o!!p(nn o& nmskp));
9     nx o|= nbp;
10    kifp(nx o< nmp) p{
11      nup[nxp] o= l+m+mi1 o% npp;
12    p}kelse p{
13      kforp(k+ktint nip(l+m+mi0p); ni o< nmp; nio++p) p{
14        kforp(k+ktint njp(l+m+mi0p), ntp(ni o+ nbp); nj o< nmp; njo++p, nto++p) p{
15          nup[ntp] o= p(nup[ntp] o+ nvp[nip] o* nvp[njp]) o% npp;
16        p}
17      p}
18      kforp(k+ktint nip((nm o<< l+m+mi1p) o- l+m+mi1p); ni o>= nmp; nio--p) p{
19        kforp(k+ktint njp(l+m+mi0p), ntp(ni o- nmp); nj o< nmp; njo++p, nto++p) p{
20          nup[ntp] o= p(nup[ntp] o+ ncp[njp] o* nup[nip]) o% npp;
21        p}
22      p}
23    p}
24    ncocyp(nup, nu o+ nmp, nvp);
25  p}
26  c+c1//a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
27  kforp(k+ktint nip(nmp); ni o< l+m+mi2 o* nmp; nio++p) p{
28    nap[nip] o= l+m+mi0p;
29    kforp(k+ktint njp(l+m+mi0p); nj o< nmp; njo++p) p{
30      nap[nip] o= p(nap[nip] o+ p(k+ktlong k+ktlongp)ncp[njp] o* nap[ni o+ nj o- nmp]) o%
  ↳ npp;
31    p}
32  p}
33  kforp(k+ktint njp(l+m+mi0p); nj o< nmp; njo++p) p{
34    nbp[njp] o= l+m+mi0p;
35    kforp(k+ktint nip(l+m+mi0p); ni o< nmp; nio++p) p{
36      nbp[njp] o= p(nbp[njp] o+ nvp[nip] o* nap[ni o+ njp]) o% npp;
37    p}
38  p}
39  kforp(k+ktint njp(l+m+mi0p); nj o< nmp; njo++p) p{
40    nap[njp] o= nbp[njp];
41  p}
42 p}

```

求逆元

```

1 k+ktvoid n+nfex_gcdp(k+ktlong k+ktlong nap, k+ktlong k+ktlong nbp, k+ktlong k+ktlong o&nxp,
  ↳ k+ktlong k+ktlong o&nyp) p{
2   kif p(nb o== l+m+mi0p) p{
3     nx o= l+m+mi1p;
4     ny o= l+m+mi0p;
5     kreturnp;
6   p}
7   k+ktlong k+ktlong nxsp, nyyp;

```

```

8   nex_gcdp(nbp, na o% nbp, nxsp, nyyp);
9   ny o= nxsp o- na o/ nb o* nyyp;
10  nx o= nyyp;
11 p}
12
13 k+ktlong k+ktlong n+nfinvp(k+ktlong k+ktlong nxp, k+ktlong k+ktlong nMODNp) p{
14   k+ktlong k+ktlong ninv_xp, nyp;
15   nex_gcdp(nxp, nMODNp, ninv_xp, nyp);
16   kreturn p(ninv_x o% nMODN o+ nMODNp) o% nMODNp;
17 p}

```

中国剩余定理

```

1 c+c1// 返回 (ans, M), 其中 ans 是模 M 意义下的解
2 nstdo::npairp<k+ktlong k+ktlongp, k+ktlong k+ktlongo> nCRTp(kconst nstdo::nvectoro<k+ktlong
  ↳ k+ktlongo>& nmp, kconst nstdo::nvectoro<k+ktlong k+ktlongo>& nap) p{
3   k+ktlong k+ktlong nM o= l+m+mi1p, nans o= l+m+mi0p;
4   k+ktint nn o= nmp.nsizep();
5   kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) nM o*= nmp[nip];
6   kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
7     nans o= p(nans o+ p(nM o/ nmp[nip]) o* nap[nip] o% nM o* ninvp(nM o/ nmp[nip]))
  ↳ o% nMp; c+c1// 可能需要大整数相乘取模
8   p}
9   kreturn nstdo::nmake_pairp(nansp, nMp);
10 p}
11 c+c1// 模数不互质的情况
12 k+ktbool nsolvep(k+ktint nnp, nstdo::npairp<k+ktlong k+ktlongp, k+ktlong k+ktlongo>
  ↳ ninputp[],
13
14   nstdo::npairp<k+ktlong k+ktlongp, k+ktlong k+ktlongo> o&noutputp) p{
15   noutput o= nstdo::nmake_pairp(l+m+mi1p, l+m+mi1p);
16   kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; o++nip) p{
17     k+ktlong k+ktlong nnumberp, nuselessp;
18     c+c1// euclid(a, b, x, y)
19     neuclidp(noutputp.nsecondp, ninputp[nip].nsecondp, nnumberp, nuselessp);
20     k+ktlong k+ktlong ndivisor o= nstdo::n_gcdp(noutputp.nsecondp, ninputp[nip].nsecondp);
21     kif p((ninputp[nip].nfirst o- noutputp.nfirstp) o% ndivisorp) kreturn n+nbfalsep;
22     nnumber o*= p(ninputp[nip].nfirst o- noutputp.nfirstp) o/ ndivisorp;
23     nfixp(nnumberp, ninputp[nip].nsecondp); c+c1// fix 成正的
24     noutputp.nfirst o+= noutputp.nsecond o* nnumberp;
25     noutputp.nsecond o*= ninputp[nip].nsecond o/ ndivisorp;
26     nfixp(noutputp.nfirstp, noutputp.nsecondp);
27   p}
28   kreturn n+nbtruep;
29 p}

```

魔法 CRT

```

1 c+c1// MOD is the given module
2 c+c1// Do not depend on LL * LL % LL
3 k+krinline k+ktint n+nfCRTp(k+ktint o*nap) p{
4   kstatic k+ktint nxp[nNp];
5   kfor p(k+ktint ni o= l+m+mi0p; ni o< nNp; ni o++p) p{
6     nxp[nip] o= nap[nip];
7     kfor p(k+ktint nj o= l+m+mi0p; nj o< nip; nj o++p) p{
8       k+ktint nt o= p(nxp[nip] o- nxp[njp] o+ nmodp[nip]) o% nmodp[nip];

```



```
9      kif p(nt o< l+m+mi0p) nt o+= nmodp[nip];
10      nxp[nip] o= l+m+mi1LL o* nt o* nInvp[njp][nip] o% nmodp[nip];
11  }
12  p}
13  k+ktint nsum o= l+m+mi1p, nret o= nxp[l+m+mi0p] o% nMODp;
14  kfor p(k+ktint ni o= l+m+mi1p; ni o< nNp; ni o++p) p{
15      nsum o= l+m+mi1LL o* nsum o* nmodp[ni o- l+m+mi1p] o% nMODp;
16      nret o+= l+m+mi1LL o* nxp[nip] o* nsum o% nMODp;
17      kif p(nret o>= nMODp) nret o-= nMODp;
18  }
19  kreturn nretp;
20  p}
21  kfor p(k+ktint ni o= l+m+mi0p; ni o< nNp; ni o++p)
22      kfor p(k+ktint nj o= ni o+ l+m+mi1p; nj o< nNp; nj o++p) p{
23          nInvp[nip][njp] o= nfpwp(nmodp[nip], nmodp[njp] o- l+m+mi2p, nmodp[njp]);
24      p}
```

素性测试

```
1  k+ktint n+nfstong_pseudo_primetestp(k+ktlong k+ktlong nnp,k+ktint nbasep) p{
2      k+ktlong k+ktlong nn2o=nno-l+m+mi1p,nresp;
3      k+ktint nso=l+m+mi0p;
4      kwhilep(nn2o%l+m+mi2o==l+m+mi0p) nn2o>=l+m+mi1p,nso++p;
5      nreso=npowmodp(nbasep,nn2p,nnp);
6      kifp((nreso==l+m+mi1p)o|p(nreso==nno-l+m+mi1p)) kreturn l+m+mi1p;
7      nso--p;
8      kwhilep(nso>=l+m+mi0p) p{
9          nreso=nmulmodp(nresp,nresp,nnp);
10         kifp(nreso==nno-l+m+mi1p) kreturn l+m+mi1p;
11         nso--p;
12     }
13     kreturn l+m+mi0p; c+c1// n is not a strong pseudo prime
14 p}
15 k+ktint n+nfisprimep(k+ktlong k+ktlong nnp) p{
16     kstatic nLL
17     ↪ ntestNump[]o=p{l+m+mi2p,l+m+mi3p,l+m+mi5p,l+m+mi7p,l+m+mi11p,l+m+mi13p,l+m+mi17p,l+m+mi19p,l+m+mi23p,l+m+mi29p,l+m+mi31p,l+m+mi37p};
18     kstatic nLL
19     ↪ nlimp[]o=p{l+m+mi4p,l+m+mi0p,l+m+mi1373653LLp,l+m+mi25326001LLp,l+m+mi25000000000LLp,l+m+mi2152301298747LLp,
20     ↪ l+m+mi3474749660383LLp,l+m+mi341550071728321LLp,l+m+mi0p,l+m+mi0p,l+m+mi0p,l+m+mi0p};
21     kifp(nno<l+m+mi2o|nno==l+m+mi3215031751LLp) kreturn l+m+mi0p;
22     kforp(k+ktint nio=l+m+mi0p;nio<l+m+mi12p;o++nip){
23         kifp(nno<nlimp[nip]) kreturn l+m+mi1p;
24         kifp(nstrong_pseudo_primetestp(nnp,ntestNump[nip])o==l+m+mi0p) kreturn l+m+mi0p;
25     }
26     kreturn l+m+mi1p;
27 p}
```

质因数分解

```
1  k+ktint nansnp; nLL nansp[l+m+mi1000p];
2  nLL n+nffuncp(nLL nxp,nLL nnp){ kreturnp(nmod_mulp(nxp,nxp,nnp)o+l+m+mi1p)o%nnp; p}
3  nLL n+nfpollardp(nLL nnp){
4      nLL nip,nxp,nyp,nnp;
5      kifp(nRabin_Millerp(nnp)) kreturn nnp;
6      kifp(o!p(nno&l+m+mi1p)) kreturn l+m+mi2p;
7      kforp(nio=l+m+mi1p;nio<l+m+mi20p;nio++p){
```

```
8      nxo=nip; nyo=nfuncp(nxp,nnp); npo=ngcdp(nyo-nxp,nnp);
9      kwhilep(npo==l+m+mi1p) p{nxo=nfuncp(nxp,nnp); nyo=nfuncp(nfuncp(nyp,nnp),nnp);
    ↪ npo=ngcdp((nyo-nxo+nnp)o%nnp,nnp)o%nnp;}
10     kifp(npo==l+m+mi0o|npo==nnp) kcontinuep;
11     kreturn npp;
12 }
13 p}
14 k+ktvoid n+nffactorp(nLL nnp){
15     nLL nxp;
16     nxo=nPollardp(nnp);
17     kifp(nxo==nnp){ nansp[nansno++p]o=nxp; kreturnp; p}
18     nfactorp(nxp), nfactorp(nno/nxp);
19 p}
```

线下整点

```
1  c+c1// ∑_{i=0}^{n-1} ⌊\frac{a+bi}{m}\rceil, n,m,a,b>0
2  nLL n+nfsolvep(nLL nnp,nLL nap,nLL nbp,nLL nmp){
3      kifp(nbo==l+m+mi0p) kreturn nno*p(nao/nmp);
4      kifp(nao>=nmp) kreturn nno*p(nao/nmp)o+nsolvep(nnp,nao%nmp,nbp,nmp);
5      kifp(nbo>=nmp) kreturn
    ↪ p(nno-l+m+mi1p)o*nno/l+m+mi2o*p(nbo/nmp)o+nsolvep(nnp,nap,nbo%nmp,nmp);
6      kreturn nsolvep((nao+nbo*nnp)o/nmp,(nao+nbo*nnp)o%nmp,nmp,nbp);
7  p}
```

原根相关

- 模 m 有原根的充要条件: $m = 2, 4, p^a, 2p^a$, 其中 p 是奇素数;
- 求任意数 p 原根的方法: 对 $\phi(p)$ 因式分解, 即 $\phi(p) = p_1^{r_1} p_2^{r_2} \cdots p_k^{r_k}$, 若恒成立:

$$g^{\frac{p-1}{q}} \not\equiv 1 \pmod{p}$$

那么 g 就是 p 的原根。

- 若模 m 有原根, 那么它一共有 $\Phi(\Phi(m))$ 个原根。

代数

快速傅里叶变换

```
1  k+ktint n+nfppreparep(k+ktint nnp) p{
2      k+ktint nlen o= l+m+mi1p;
3      kfor p(; nlen o<= l+m+mi2 o* nnp; nlen o<= l+m+mi1p);
4      kfor p(k+ktint ni o= l+m+mi0p; ni o< nlenp; nio++p) p{
5          nep[l+m+mi0p][nip] o= nComplexp(ncosp(l+m+mi2 o* npi o* ni o/ nlenp), nsinp(l+m+mi2 o*
    ↪ npi o* ni o/ nlenp));
6          nep[l+m+mi1p][nip] o= nComplexp(ncosp(l+m+mi2 o* npi o* ni o/ nlenp), o-nsinp(l+m+mi2 o*
    ↪ npi o* ni o/ nlenp));
7      }
8      kreturn nlenp;
9  p}
10 k+ktvoid n+nfdFTp(nComplex o*nap, k+ktint nnp, k+ktint nfp) p{
11     kfor p(k+ktint ni o= l+m+mi0p, nj o= l+m+mi0p; ni o< nnp; nio++p) p{
12         kif p(ni o> njp) nstdo::nswapp(nap[nip], nap[njp]);
13         kfor p(k+ktint nt o= nn o>> l+m+mi1p; p(nj o^= ntp) o< ntp; nt o>= l+m+mi1p);
14     p}
```

```

15 kfor p(k+ktint ni o= l+m+mi2p; ni o<= nnp; ni o<= l+m+mi1p)
16   kfor p(k+ktint nj o= l+m+mi0p; nj o< nnp; nj o+= nip)
17     kfor p(k+ktint nk o= l+m+mi0p; nk o< p(ni o>> l+m+mi1p); nko++p) p{
18       nComplex nA o= nap[nj o+ nkp];
19       nComplex nB o= nep[nfp][nn o/ ni o* nkp] o* nap[nj o+ nk o+ p(ni o>> l+m+mi1p)];
20       nap[nj o+ nkp] o= nA o+ nBp;
21       nap[nj o+ nk o+ p(ni o>> l+m+mi1p)] o= nA o- nBp;
22     p}
23   kif p(nf o== l+m+mi1p) p{
24     kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p)
25       nap[nip].na o/= nnp;
26   p}
27 p}

```

快速数论变换

```

1 c+c1// meminit(A, l, r) 是将数组 A 的 [l, r) 清 0。
2 c+c1// memcpy(target, source, l, r) 是将 source 的 [l, r) 复制到 target 的 [l, r)
3 c+cp#define meminit(A, l, r) memset(A + (l), 0, sizeof(*A) * ((r) - (l)))
4 c+cp#define memcpy(B, A, l, r) memcpy(B, A + (l), sizeof(*A) * ((r) - (l)))
5 k+ktvoid n+nfdFTp(k+ktint o*nap, k+ktint nnp, k+ktint nfp) p{ c+c1// 封闭形式, 常数小
  ↪ (107 跑 2.23 秒)
6   kfor p(kregister k+ktint ni o= l+m+mi0p, nj o= l+m+mi0p; ni o< nnp; nio++p) p{
7     kif p(ni o> njp) nstdo::nswapp(nap[nip], nap[njp]);
8     kfor p(kregister k+ktint nt o= nn o>> l+m+mi1p; p(nj o^= ntp) o< ntp; nt o>>= l+m+mi1p);
9   p}
10  kfor p(kregister k+ktint ni o= l+m+mi2p; ni o<= nnp; ni o<= l+m+mi1p) p{
11    kstatic k+ktint nexpp[nMAXNp];
12    nexpp[l+m+mi0p] o= l+m+mi1p; nexpp[l+m+mi1p] o= nfpmp(nPRTp, p(nMOD o- l+m+mi1p) o/ nip);
13    kif p(nf o== l+m+mi1p) nexpp[l+m+mi1p] o= nfpmp(nexpp[l+m+mi1p], nMOD o- l+m+mi2p);
14    kfor p(kregister k+ktint nk o= l+m+mi2p; nk o< p(ni o>> l+m+mi1p); nko++p) p{
15      nexpp[nkp] o= l+m+mi1ll o* nexpp[nk o- l+m+mi1p] o* nexpp[l+m+mi1p] o% nMODp;
16    p}
17    kfor p(kregister k+ktint nj o= l+m+mi0p; nj o< nnp; nj o+= nip) p{
18      kfor p(kregister k+ktint nk o= l+m+mi0p; nk o< p(ni o>> l+m+mi1p); nko++p) p{
19        kregister k+ktint o&nPA o= nap[nj o+ nkp], o&nPB o= nap[nj o+ nk o+ p(ni o>>
  ↪ l+m+mi1p)];
20        kregister k+ktint nA o= npAp, nB o= l+m+mi1ll o* npB o* nexpp[nkp] o% nMODp;
21        npA o= p(nA o+ nBp) o% nMODp;
22        npB o= p(nA o- nB o+ nMODp) o% nMODp;
23      p}
24    p}
25  p}
26  kif p(nf o== l+m+mi1p) p{
27    kregister k+ktint nrev o= nfpmp(nnp, nMOD o- l+m+mi2p, nMODp);
28    kfor p(kregister k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
29      nap[nip] o= l+m+mi1ll o* nap[nip] o* nrev o% nMODp;
30    p}
31  p}
32 p}
33 c+c1// 在不写高精度的情况下合并 FFT 所得结果对 MOD 取模后的答案
34 c+c1// 值得注意的是, 这个东西不能最后再合并, 而是应该每做一次多项式乘法就 CRT
  ↪ 一次
35 k+ktint n+nfcRTp(k+ktint o*nap) p{
36   kstatic k+ktint nxp[l+m+mi3p];

```

```

37 kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi3p; nio++p) p{
38   nxp[nip] o= nap[nip];
39   kfor p(k+ktint nj o= l+m+mi0p; nj o< nip; njo++p) p{
40     k+ktint nt o= p(nxp[nip] o- nxp[njp] o+ nFFTp[nip] o-> nMODp) o% nFFTp[nip] o-> nMODp;
41     kif p(nt o< l+m+mi0p) nt o+= nFFTp[nip] o-> nMODp;
42     nxp[nip] o= l+m+mi1LL o* nt o* ninvp[njp][nip] o% nFFTp[nip] o-> nMODp;
43   p}
44 p}
45 k+ktint nsum o= l+m+mi1p, nret o= nxp[l+m+mi0p] o% nMODp;
46 kfor p(k+ktint ni o= l+m+mi1p; ni o< l+m+mi3p; ni o++p) p{
47   nsum o= l+m+mi1LL o* nsum o* nFFTp[ni o- l+m+mi1p] o-> nMOD o% nMODp;
48   nret o+= l+m+mi1LL o* nxp[nip] o* nsum o% nMODp;
49   kifp(nret o>= nMODp) nret o-= nMODp;
50 p}
51 kreturn nret;
52 p}
53 kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi3p; nio++p) c+c1// inv 数组的预处理过程,
  ↪ inverse(x, p) 表示求 x 在 p 下逆元
54   kfor p(k+ktint nj o= l+m+mi0p; nj o< l+m+mi3p; njo++p)
55     ninvp[nip][njp] o= ninversep(nFFTp[nip] o-> nMODp, nFFTp[njp] o-> nMODp);

```

自适应辛普森积分

```

1 namespace nadaptive_simpson p{
2   ktemplateo<typename nfunctiono>
3   k+krinline k+ktdouble nareap(nfunction nfp, kconst k+ktdouble o&nleftp, kconst k+ktdouble
  ↪ o&nrightp) p{
4     k+ktdouble nmid o= p(nleftp o+ nrightp) o/ l+m+mi2p;
5     kreturn p(nright o- nleftp) o* p(nfp(nleftp) o+ l+m+mi4 o* nfp(nmidp) o+ nfp(nrightp)) o/
  ↪ l+m+mi6p;
6   p}
7
8   ktemplateo<typename nfunctiono>
9   k+krinline k+ktdouble nsimpsonp(nfunction nfp, kconst k+ktdouble o&nleftp, kconst
  ↪ k+ktdouble o&nrightp, kconst k+ktdouble o&nepsp, kconst k+ktdouble o&narea_ump) p{
10    k+ktdouble nmid o= p(nleftp o+ nrightp) o/ l+m+mi2p;
11    k+ktdouble narea_left o= nareap(nfp, nleftp, nmidp);
12    k+ktdouble narea_right o= nareap(nfp, nmidp, nrightp);
13    k+ktdouble narea_total o= narea_left o+ narea_rightp;
14    kif p(nfabsp(narea_total o- narea_ump) o<= l+m+mi15 o* nepsp) p{
15      kreturn narea_total o+ p(narea_total o- narea_ump) o/ l+m+mi15p;
16    p}
17    kreturn nsimpsonp(nfp, nleftp, nrightp, nepsp o/ l+m+mi2p, narea_leftp) o+ nsimpsonp(nfp,
  ↪ nmidp, nrightp, nepsp o/ l+m+mi2p, narea_rightp);
18  p}
19
20   ktemplateo<typename nfunctiono>
21   k+krinline k+ktdouble nsimpsonp(nfunction nfp, kconst k+ktdouble o&nleftp, kconst
  ↪ k+ktdouble o&nrightp, kconst k+ktdouble o&nepsp) p{
22     kreturn nsimpsonp(nfp, nleftp, nrightp, nepsp, nareap(nfp, nleftp, nrightp));
23   p}
24 p}

```

单纯形

```

1 kconst k+ktdouble neps o= l+m+mf1e-8p;
2 c+c1// max{c * x | Ax <= b, x >= 0} 的解, 无解返回空的 vector, 否则就是解.
3 nvectoro<k+ktdoubleo> nsimplexp(nvectoro<nvectoro<k+ktdoubleo> o> o&nAp,
   ↪ nvectoro<k+ktdoubleo> nbp, nvectoro<k+ktdoubleo> ncp) p{
4 k+ktint nn o= nAp.nsizep(), nm o= nAp[l+m+mi0p].nsizep() o+ l+m+mi1p, nr o= nnp, ns o= nm
   ↪ o- l+m+mi1p;
5 nvectoro<nvectoro<k+ktdoubleo> o> nDp(nn o+ l+m+mi2p, nvectoro<k+ktdoubleo>p(nm o+
   ↪ l+m+mi1p));
6 nvectoro<k+ktinto> nixp(nn o+ nmp);
7 kforp(k+ktint ni o= l+m+mi0p; ni o< nn o+ nmp; nio++p) p{
8 nixp[nip] o= nip;
9 p}
10 kforp(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
11 kforp(k+ktint nj o= l+m+mi0p; nj o< nm o- l+m+mi1p; njo++p) p{
12 nDp[nip][njp] o= o-nAp[nip][njp];
13 p}
14 nDp[nip][nm o- l+m+mi1p] o= l+m+mi1p;
15 nDp[nip][nmp] o= nbp[nip];
16 kif p(nDp[nrp][nmp] o> nDp[nip][nmp]) p{
17 nr o= nip;
18 p}
19 p}
20
21 kforp(k+ktint nj o= l+m+mi0p; nj o< nm o- l+m+mi1p; njo++p) p{
22 nDp[nnp][njp] o= ncp[njp];
23 p}
24 nDp[nn o+ l+m+mi1p][nm o- l+m+mi1p] o= o-l+m+mi1p;
25 kforp(k+ktdouble ndp; p;) p{
26 kif p(nr o< nnp) p{
27 nswapp(nixp[nsp], nixp[nr o+ nmp]);
28 nDp[nrp][nsp] o= l+m+mf1. o/ nDp[nrp][nsp];
29 kforp(k+ktint nj o= l+m+mi0p; nj o<= nmp; njo++p) p{
30 kif p(nj o!= nsp) p{
31 nDp[nrp][njp] o*= o-nDp[nrp][nsp];
32 p}
33 p}
34 kforp(k+ktint ni o= l+m+mi0p; ni o<= nn o+ l+m+mi1p; nio++p) p{
35 kif p(ni o!= nrp) p{
36 kforp(k+ktint nj o= l+m+mi0p; nj o<= nmp; njo++p) p{
37 kif p(nj o!= nsp) p{
38 nDp[nip][njp] o+= nDp[nrp][njp] o* nDp[nip][nsp];
39 p}
40 p}
41 nDp[nip][nsp] o*= nDp[nrp][nsp];
42 p}
43 p}
44 p}
45 nr o= o-l+m+mi1p, ns o= o-l+m+mi1p;
46 kforp(k+ktint nj o= l+m+mi0p; nj o< nmp; njo++p) p{
47 kif p(ns o< l+m+mi0 o|| nixp[nsp] o> nixp[njp]) p{
48 kif p(nDp[nn o+ l+m+mi1p][njp] o> neps o|| nDp[nn o+ l+m+mi1p][njp] o> o-neps o&&
   ↪ nDp[nnp][njp] o> neps) p{
49 ns o= njp;
50 p}

```

```

51 p}
52 p}
53 kif p(ns o< l+m+mi0p) p{
54 kbreakp;
55 p}
56 kforp(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
57 kif p(nDp[nip][nsp] o< o-neps) p{
58 kif p(nr o< l+m+mi0 o|| p(nd o= nDp[nrp][nmp] o/ nDp[nrp][nsp] o- nDp[nip][nmp] o/
   ↪ nDp[nip][nsp]) o< o-neps
   o|| nd o< neps o&& nixp[nr o+ nmp] o> nixp[ni o+ nmp]) p{
59
60
61 nr o= nip;
62 p}
63 p}
64 p}
65
66 kif p(nr o< l+m+mi0p) p{
67 kreturn nvectoro<k+ktdoubleo> p();
68 p}
69 p}
70 kif p(nDp[nn o+ l+m+mi1p][nmp] o< o-neps) p{
71 kreturn nvectoro<k+ktdoubleo> p();
72 p}
73
74 nvectoro<k+ktdoubleo> nxp(nm o- l+m+mi1p);
75 kforp(k+ktint ni o= nmp; ni o< nn o+ nmp; nio++p) p{
76 kif p(nixp[nip] o< nm o- l+m+mi1p) p{
77 nxp[nixp[nip]] o= nDp[ni o- nmp][nmp];
78 p}
79 p}
80 kreturn nxp;
81 p}

```

字符串

后缀数组

```

1 kconst k+ktint nMAXN o= nMAXL o* l+m+mi2 o+ l+m+mi1p;
2 k+ktint nap[nMAXNp], nxp[nMAXNp], nyp[nMAXNp], ncp[nMAXNp], nsap[nMAXNp], nrankp[nMAXNp],
   ↪ nheightp[nMAXNp];
3 k+ktvoid n+nfcalsap(k+ktint nnp) p{
4 k+ktint nm o= nalphabetp, nk o= l+m+mi1p;
5 nmemsetp(ncp, l+m+mi0p, ksizeofp(o*ncp) o* p(nm o+ l+m+mi1p));
6 kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) ncp[nxp[nip] o= nap[nip]]o++p;
7 kfor p(k+ktint ni o= l+m+mi1p; ni o<= nmp; o++nip) ncp[nip] o+= ncp[ni o- l+m+mi1p];
8 kfor p(k+ktint ni o= nnp; nip; o--nip) nsap[ncp[nxp[nip]]o--p] o= nip;
9 kfor p(; nk o<= nnp; nk o<= l+m+mi1p) p{
10 k+ktint ntot o= nkp;
11 kfor p(k+ktint ni o= nn o- nk o+ l+m+mi1p; ni o<= nnp; o++nip) nyp[ni o- nn o+ nkp] o=
   ↪ nip;
12 kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip)
13 kif p(nsap[nip] o> nkp) nyp[o++ntotp] o= nsap[nip] o- nkp;
14 nmemsetp(ncp, l+m+mi0p, ksizeofp(o*ncp) o* p(nm o+ l+m+mi1p));
15 kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) ncp[nxp[nip]]o++p;
16 kfor p(k+ktint ni o= l+m+mi1p; ni o<= nmp; o++nip) ncp[nip] o+= ncp[ni o- l+m+mi1p];

```

```

17     kfor p(k+ktint ni o= nnp; nlp; o--nlp) nsap[ncp[nxp[nyp[nip]]]o--p] o= nyp[nip];
18     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nlp) nyp[nip] o= nxp[nip];
19     ntot o= l+m+mi1p; nxp[nsap[l+m+mi1p]] o= l+m+mi1p;
20     kfor p(k+ktint ni o= l+m+mi2p; ni o<= nnp; o++nlp) p{
21         kif p(nmaxp(nsap[nip], nsap[ni o- l+m+mi1p]) o+ nk o> nn o|| nyp[nsap[nip]] o!=
↪ nyp[nsap[ni o- l+m+mi1p]] o|| nyp[nsap[nip] o+ nkp] o!= nyp[nsap[ni o- l+m+mi1p] o+
↪ nkp]) o++ntotp;
22         nxp[nsap[nip]] o= ntotp;
23     p}
24     kif p(ntot o== nnp) kbreakp; kelse nm o= ntotp;
25 p}
26 p}
27 k+ktvoid n+nfcalc_heightp(k+ktint nnp) p{
28     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nlp) nrankp[nsap[nip]] o= nlp;
29     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nlp) p{
30         nheightp[nrankp[nip]] o= nmaxp(l+m+mi0p, nheightp[nrankp[ni o- l+m+mi1p]] o- l+m+mi1p);
31         kif p(nrankp[nip] o== l+m+mi1p) kcontinuep;
32         k+ktint nj o= nsap[nrankp[nip] o- l+m+mi1p];
33         kwhile p(nmaxp(nip, njp) o+ nheightp[nrankp[nip]] o<= nn o&& nap[ni o+
↪ nheightp[nrankp[nip]]] o== nap[nj o+ nheightp[nrankp[nip]]]) o++nheightp[nrankp[nip]];
34     p}
35 p}

```

后缀自动机

```

1 kstatic kconst k+ktint nMAXL o= nMAXN o* l+m+mi2p; c+c1// MAXN is original length
2 kstatic kconst k+ktint na1phabet o= l+m+mi26p; c+c1// sometimes need changing
3 k+ktint nlp, nlastp, ncntp, ntransp[nMAXLp][na1phabetp], nparp[nMAXLp], nsump[nMAXLp],
↪ nseqp[nMAXLp], nm1lp[nMAXLp], nsizep[nMAXLp]; c+c1// mxl is maxlength, size is the size
↪ of right
4 k+ktchar nstrp[nMAXLp];
5 k+krinline k+ktvoid n+nfinitp() p{
6     nl o= nstrlenp(nstr o+ l+m+mi1p); ncntp o= nlast o= l+m+mi1p;
7     kfor p(k+ktint ni o= l+m+mi0p; ni o<= nl o* l+m+mi2p; o++nlp) nmemsetp(ntransp[nip],
↪ l+m+mi0p, ksizeofp(ntransp[nip]));
8     nmemsetp(nparp, l+m+mi0p, ksizeofp(o*nparp) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
9     nmemsetp(nm1lp, l+m+mi0p, ksizeofp(o*nm1lp) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
10    nmemsetp(nsizep, l+m+mi0p, ksizeofp(o*nsizep) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
11 p}
12 k+krinline k+ktvoid n+nfextendp(k+ktint nposp, k+ktint ncp) p{
13     k+ktint np o= nlastp, nnp o= nlast o= o++ncntp;
14     nm1lp[nnp] o= nm1lp[npp] o+ l+m+mi1p; nsizep[nnp] o= l+m+mi1p;
15     kfor p(; np o&& o!ntransp[npp][ncp]; np o= nparp[npp]) ntransp[npp][ncp] o= nnp;
16     kif p(o!npp) nparp[npp] o= l+m+mi1p;
17     kelse p{
18         k+ktint nq o= ntransp[npp][ncp];
19         kif p(nm1lp[npp] o+ l+m+mi1 o== nm1lp[nqp]) nparp[npp] o= nqp;
20         kelse p{
21             k+ktint nnq o= o++ncntp;
22             nm1lp[nnq] o= nm1lp[npp] o+ l+m+mi1p;
23             nmemcpyp(ntransp[nnq], ntransp[nqp], ksizeofp(ntransp[nnq]));
24             nparp[nnq] o= nparp[nqp];
25             nparp[npp] o= nparp[nqp] o= nnq;
26             kfor p(; ntransp[npp][ncp] o== nqp; np o= nparp[npp]) ntransp[npp][ncp] o= nnq;
27         p}
28 p}

```

```

29 p}
30 k+krinline k+ktvoid n+nfbuildsmp() p{
31     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nlp; o++nlp) nextendp(nip, nstrp[nip] o- l+s+sc'a'p);
32     nmemsetp(nsump, l+m+mi0p, ksizeofp(o*nsump) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
33     kfor p(k+ktint ni o= l+m+mi1p; ni o<= ncntp; o++nlp) nsump[nm1lp[nip]]o++p;
34     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nlp; o++nlp) nsump[nip] o+= nsump[ni o- l+m+mi1p];
35     kfor p(k+ktint ni o= ncntp; nlp; o--nlp) nseqp[nsump[nm1lp[nip]]o--p] o= nip;
36     kfor p(k+ktint ni o= ncntp; nlp; o--nlp) nsizep[nparp[nseqp[nip]]] o+= nsizep[nseqp[nip]];
37 p}

```

EX 后缀自动机

```

1 k+krinline k+ktvoid n+nfadd_nodep(k+ktint nxp, k+ktint o&nlastp) p{
2     k+ktint nlastnode o= nlastp;
3     kif p(ncp[nlastnode][nxp]) p{
4         k+ktint nnewnode o= ncp[nlastnode][nxp];
5         kif p(nlp[nnewnode] o== nlp[nlastnode] o+ l+m+mi1p) nlast o= nnewnode;
6         kelse p{
7             k+ktint nauxnode o= o++ncntp; nlp[nauxnode] o= nlp[nlastnode] o+ l+m+mi1p;
8             kfor p(k+ktint ni o= l+m+mi0p; ni o< na1phabetp; o++nlp) ncp[nauxnode][nip] o=
↪ ncp[nnewnode][nip];
9             nparp[nauxnode] o= nparp[nnewnode]; nparp[nnewnode] o= nauxnode;
10            kfor p(; nlastnode o&& ncp[nlastnode][nxp] o== nnewnode; nlastnode o=
↪ nparp[nlastnode]) p{
11                ncp[nlastnode][nxp] o= nauxnode;
12            p}
13            nlast o= nauxnode;
14        p}
15    p} kelse p{
16        k+ktint nnewnode o= o++ncntp; nlp[nnewnode] o= nlp[nlastnode] o+ l+m+mi1p;
17        kfor p(; nlastnode o&& o!ncp[nlastnode][nxp]; nlastnode o= nparp[nlastnode])
↪ ncp[nlastnode][nxp] o= nnewnode;
18        kif p(o!nlastnode) nparp[nnewnode] o= l+m+mi1p;
19        kelse p{
20            k+ktint nnewnode o= ncp[nlastnode][nxp];
21            kif p(nlp[nlastnode] o+ l+m+mi1 o== nlp[nnewnode]) nparp[nnewnode] o= nnewnode;
22            kelse p{
23                k+ktint nauxnode o= o++ncntp; nlp[nauxnode] o= nlp[nlastnode] o+ l+m+mi1p;
24                kfor p(k+ktint ni o= l+m+mi0p; ni o< na1phabetp; o++nlp) ncp[nauxnode][nip] o=
↪ ncp[nnewnode][nip];
25                nparp[nauxnode] o= nparp[nnewnode]; nparp[nnewnode] o= nparp[nnewnode] o=
↪ nauxnode;
26                kfor p(; nlastnode o&& ncp[nlastnode][nxp] o== nnewnode; nlastnode o=
↪ nparp[nlastnode]) p{
27                    ncp[nlastnode][nxp] o= nauxnode;
28                p}
29            p}
30        p}
31        nlast o= nnewnode;
32    p}
33 p}

```

后缀树

1. 边上的字符区间是左闭右开区间；

2. 如果要建立关于多个串的后缀树，请用不同的分隔符，并且对于每个叶子结点，去掉和它父亲的连边上出现的第一个分隔符之后的所有字符；

回文自动机

```

1 k+ktint nnTp, nnStrp, nlastp, ncp[nMAXTp][l+m+mi26p], nfailp[nMAXTp], nrp[nMAXNp],
  ↳ nlp[nMAXNp], nsp[nMAXNp];
2 k+ktint n+nfallocatep(k+ktint nlenp) p{
3   nlp[nnTp] o= nlenp;
4   nrp[nnTp] o= l+m+mi0p;
5   nfailp[nnTp] o= l+m+mi0p;
6   nmemsetp(ncp[nnTp], l+m+mi0p, ksizeofp(ncp[nnTp]));
7   kreturn nnTo++p;
8 p}
9 k+ktvoid n+nfinitp() p{
10  nnT o= nnStr o= l+m+mi0p;
11  k+ktint nnewE o= nallocatep(l+m+mi0p);
12  k+ktint nnewO o= nallocatep(o-l+m+mi1p);
13  nlast o= nnewEp;
14  nfailp[nnewEp] o= nnewOp;
15  nfailp[nnewOp] o= nnewEp;
16  nsp[l+m+mi0p] o= o-l+m+mi1p;
17 p}
18 k+ktvoid n+nfaddp(k+ktint nxp) p{
19  nsp[o++nnStrp] o= nxp;
20  k+ktint nnow o= nlastp;
21  kwhile p(nsp[nnStr o- nlp[nnowp] o- l+m+mi1p] o!= nsp[nnStrp]) nnow o= nfailp[nnowp];
22  kif p(o!ncp[nnowp][nxp]) p{
23    k+ktint nnewnode o= nallocatep(nlp[nnowp] o+ l+m+mi2p), o&nnewfail o= nfailp[nnewnodep];
24    nnewfail o= nfailp[nnowp];
25    kwhile p(nsp[nnStr o- nlp[nnewfailp] o- l+m+mi1p] o!= nsp[nnStrp]) nnewfail o=
      ↳ nfailp[nnewfailp];
26    nnewfail o= ncp[nnewfailp][nxp];
27    ncp[nnowp][nxp] o= nnewnodep;
28  p}
29  nlast o= ncp[nnowp][nxp];
30  nrp[nlastp]o++p;
31 p}
32 k+ktvoid n+nfcountp() p{
33  kfor p(k+ktint ni o= nnT o- l+m+mi1p; ni o>= l+m+mi0p; nio--p) p{
34    nrp[nfailp[nip]] o+= nrp[nip];
35  p}
36 p}

```

数据结构

KD-Tree

```

1 k+ktlong k+ktlong n+nfnormp(kconst k+ktlong k+ktlong o&nxp) p{
2   c+c1//    For manhattan distance
3   kreturn nstdo::nabsp(nxp);
4   c+c1//    For euclid distance
5   kreturn nx o* nxp;
6 p}
7
8 kstruct nPoint p{
9   k+ktint nxp, nyp, nidp;

```

```

10
11 kconst k+ktint& koperator p[] p(k+ktint nindexp) kconst p{
12   kif p(nindex o== l+m+mi0p) p{
13     kreturn nxp;
14   p} kelse p{
15     kreturn nyp;
16   p}
17 p}
18
19 kfriend k+ktlong k+ktlong ndistp(kconst nPoint o&nap, kconst nPoint o&nbp) p{
20   k+ktlong k+ktlong nresult o= l+m+mi0p;
21   kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; o++nip) p{
22     nresult o+= nnormp(nap[nip] o- nbp[nip]);
23   p}
24   kreturn nresultp;
25 p}
26 p} npointp[nNp];
27
28 kstruct nRectangle p{
29   k+ktint nminp[l+m+mi2p], nmaxp[l+m+mi2p];
30
31 nRectanglep() p{
32   nminp[l+m+mi0p] o= nminp[l+m+mi1p] o= nINT_MAXp; c+c1// sometimes int is not enough
33   nmaxp[l+m+mi0p] o= nmaxp[l+m+mi1p] o= nINT_MINp;
34 p}
35
36 k+ktvoid naddp(kconst nPoint o&npp) p{
37   kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; o++nip) p{
38     nminp[nip] o= nstdo::nminp(nminp[nip], npp[nip]);
39     nmaxp[nip] o= nstdo::nmaxp(nmaxp[nip], npp[nip]);
40   p}
41 p}
42
43 k+ktlong k+ktlong ndistp(kconst nPoint o&npp) p{
44   k+ktlong k+ktlong nresult o= l+m+mi0p;
45   kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; o++nip) p{
46     c+c1//    For minimum distance
47     nresult o+= nnormp(nstdo::nminp(nstdo::nmaxp(npp[nip], nminp[nip]), nmaxp[nip])
↳ o- npp[nip]);
48     c+c1//    For maximum distance
49     nresult o+= nstdo::nmaxp(nnormp(nmaxp[nip] o- npp[nip]), nnormp(nminp[nip] o-
↳ npp[nip]));
50   p}
51   kreturn nresultp;
52 p}
53 p};
54
55 kstruct nNode p{
56   nPoint nseparatorp;
57   nRectangle nrectanglep;
58   k+ktint nchildp[l+m+mi2p];
59
60 k+ktvoid n+nfresetp(kconst nPoint o&npp) p{
61   nseparator o= npp;
62   nrectangle o= nRectanglep();

```

```

63     nrectanglep.naddp(npp);
64     nchildp[l+m+mi0p] o= nchildp[l+m+mi1p] o= l+m+mi0p;
65     p}
66 p} ntreep[nN o<< l+m+mi1p];
67
68 k+ktint nsizep, npivotp;
69
70 k+ktbool n+nfcomparep(kconst nPoint o&nap, kconst nPoint o&nbp) p{
71     kif p(nap[npivotp] o!= nbp[npivotp]) p{
72         kreturn nap[npivotp] o< nbp[npivotp];
73     p}
74     kreturn nap.nid o< nbp.nidp;
75 p}
76
77 c+c1// 左閉右開: build(1, n + 1)
78 k+ktint n+nfbuildp(k+ktint nlp, k+ktint nrp, k+ktint ntype o= l+m+mi1p) p{
79     npivot o= ntypep;
80     kif p(nl o>= nrp) p{
81         kreturn l+m+mi0p;
82     p}
83     k+ktint nx o= o++nsizep;
84     k+ktint nmid o= nl o+ nr o>> l+m+mi1p;
85     nstdo::nnth_elementp(npoint o+ nlp, npoint o+ nmidp, npoint o+ nrp, ncomparep);
86     ntreep[nxp].nresetp(npointp[nmidp]);
87     kfor p(k+ktint ni o= nlp; ni o< nrp; o++nlp) p{
88         ntreep[nxp].nrectanglep.naddp(npointp[nip]);
89     p}
90     ntreep[nxp].nchildp[l+m+mi0p] o= nbuidp(nlp, nmidp, ntype o^ l+m+mi1p);
91     ntreep[nxp].nchildp[l+m+mi1p] o= nbuidp(nmid o+ l+m+mi1p, nrp, ntype o^ l+m+mi1p);
92     kreturn nxp;
93 p}
94
95 k+ktint n+nfinsertp(k+ktint nxp, kconst nPoint o&npp, k+ktint ntype o= l+m+mi1p) p{
96     npivot o= ntypep;
97     kif p(nx o== l+m+mi0p) p{
98         ntreep[o++nsizep].nresetp(npp);
99         kreturn nsizep;
100     p}
101     ntreep[nxp].nrectanglep.naddp(npp);
102     kif p(ncomparep(npp, ntreep[nxp].nseparatorp)) p{
103         ntreep[nxp].nchildp[l+m+mi0p] o= ninsertp(ntreep[nxp].nchildp[l+m+mi0p], npp, ntype
        ↪ o^ l+m+mi1p);
104     p} kelse p{
105         ntreep[nxp].nchildp[l+m+mi1p] o= ninsertp(ntreep[nxp].nchildp[l+m+mi1p], npp, ntype
        ↪ o^ l+m+mi1p);
106     p}
107     kreturn nxp;
108 p}
109
110 c+c1// For minimum distance
111 c+c1// For maximum: 下面递归 query 时 0, 1 换顺序;< and >;min and max
112 k+ktvoid n+nfqueryp(k+ktint nxp, kconst nPoint o&npp, nstdo::npairo<k+ktlong k+ktlongp,
        ↪ k+ktinto> o&nanswerp, k+ktint ntype o= l+m+mi1p) p{
113     npivot o= ntypep;
114     kif p(nx o== l+m+mi0 o|| ntreep[nxp].nrectanglep.ndistp(npp) o> nanswerp.nfirstp) p{

```

```

115     kreturnp;
116 p}
117     nanswer o= nstdo::nminp(nanswerp,
118         nstdo::nmake_pairp(ndistp(ntreep[nxp].nseparatorp, npp),
        ↪ ntreep[nxp].nseparatorp.nidp));
119     kif p(ncomparep(npp, ntreep[nxp].nseparatorp)) p{
120         nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nanswerp, ntype o^ l+m+mi1p);
121         nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nanswerp, ntype o^ l+m+mi1p);
122     p} kelse p{
123         nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nanswerp, ntype o^ l+m+mi1p);
124         nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nanswerp, ntype o^ l+m+mi1p);
125     p}
126 p}
127
128 nstdo::npriority_queueo<nstdo::npairo<k+ktlong k+ktlongp, k+ktinto> o> nanswerp;
129
130 k+ktvoid n+nfqueryp(k+ktint nxp, kconst nPoint o&npp, k+ktint nkp, k+ktint ntype o= l+m+mi1p)
        ↪ p{
131     npivot o= ntypep;
132     kif p(nx o== l+m+mi0 o|| p(k+ktintp)nanswerp.nsizep() o== nk o&&
        ↪ ntreep[nxp].nrectanglep.ndistp(npp) o> nanswerp.ntopp().nfirstp) p{
133         kreturnp;
134     p}
135     nanswerp.npushp(nstdo::nmake_pairp(ndistp(ntreep[nxp].nseparatorp, npp),
        ↪ ntreep[nxp].nseparatorp.nidp));
136     kif p((k+ktintp)nanswerp.nsizep() o> nkp) p{
137         nanswerp.npopp();
138     p}
139     kif p(ncomparep(npp, ntreep[nxp].nseparatorp)) p{
140         nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nkp, ntype o^ l+m+mi1p);
141         nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nkp, ntype o^ l+m+mi1p);
142     p} kelse p{
143         nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nkp, ntype o^ l+m+mi1p);
144         nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nkp, ntype o^ l+m+mi1p);
145     p}
146 p}

```

Treap

```

1 kstruct nNodep{
2     k+ktint nmnp, nkeyp, nsizep, ntagp;
3     k+ktbool nrevp;
4     nNodeo* nchp[l+m+mi2p];
5     nNodep(k+ktint nmnp, k+ktint nkeyp, k+ktint nsizep)o: nmnp(nmnp), nkeyp(nkeyp),
        ↪ nsizep(nsizep), nrevp(l+m+mi0p), ntagp(l+m+mi0p){}
6     k+ktvoid ndowntagp();
7     nNodeo* n+nfupdatep(){
8         nmnp o= nminp(nchp[l+m+mi0p] o-> nmnp, nminp(nkeyp, nchp[l+m+mi1p] o-> nmnp));
9         nsize o= nchp[l+m+mi0p] o-> nsize o+ l+m+mi1 o+ nchp[l+m+mi1p] o-> nsizep;
10        kreturn kthisp;
11    p}
12 p};
13 ktypedef npairo<nNodeo*p, nNodeo*> nPairp;
14 nNode o*nnullp, o*nrootp;
15 k+ktvoid nNodeo::ndowntagp(){

```

```

16 kifp(nrevp){
17     kforp(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; nio++p)
18         kifp(nchp[nip] o!= nullptr){
19             nchp[nip] o-> nrev o+= l+m+mi1p;
20             nswapp(nchp[nip] o-> nchp[l+m+mi0p], nchp[nip] o-> nchp[l+m+mi1p]);
21         p}
22     nrev o= l+m+mi0p;
23 p}
24 kifp(ntagp){
25     kforp(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; nio++p)
26         kifp(nchp[nip] o!= nullptr){
27             nchp[nip] o-> nkey o+= ntagp;
28             nchp[nip] o-> nmn o+= ntagp;
29             nchp[nip] o-> ntag o+= ntagp;
30         p}
31     ntag o= l+m+mi0p;
32 p}
33 p}
34 k+ktint nrp(){
35     kstatic k+ktint ns o= l+m+mi3023192386p;
36     kreturn p(ns o+= p(ns o<< l+m+mi3p) o+ l+m+mi1p) o& p(o~l+m+mi0u o>> l+m+mi1p);
37 p}
38 k+ktbool nrandomp(k+ktint nxp, k+ktint nyp){
39     kreturn nrp() o% p(nx o+ nyp) o< nxp;
40 p}
41 nNodeo* nmergep(nNode o*npp, nNode o*nqp){
42     kifp(np o== nullptr) kreturn nqp;
43     kifp(nq o== nullptr) kreturn npp;
44     np o-> ndowntagp();
45     nq o-> ndowntagp();
46     kifp(nrandomp(np o-> nsizep, nq o-> nsizep)){
47         np o-> nchp[l+m+mi1p] o= nmergep(np o-> nchp[l+m+mi1p], nqp);
48         kreturn np o-> nupdatep();
49 p}kelsep{
50     nq o-> nchp[l+m+mi0p] o= nmergep(npp, nq o-> nchp[l+m+mi0p]);
51     kreturn nq o-> nupdatep();
52 p}
53 p}
54 nPair nsplitp(nNode o*nxp, k+ktint nnp){
55     kifp(nx o== nullptr) kreturn nmake_pairp(nullptr, nullptr);
56     nx o-> ndowntagp();
57     kifp(nn o<= nx o-> nchp[l+m+mi0p] o-> nsizep){
58         nPair nret o= nsplitp(nx o-> nchp[l+m+mi0p], nnp);
59         nx o-> nchp[l+m+mi0p] o= nretp.nsecondp;
60         kreturn n+nfmake_pairp(nretp.nfirstp, nx o-> nupdatep());
61 p}
62 nPair nret o= nsplitp(nx o-> nchp[l+m+mi1p], nn o- nx o-> nchp[l+m+mi0p] o-> nsize o-
    ↪ l+m+mi1p);
63 nx o-> nchp[l+m+mi1p] o= nretp.nfirstp;
64 kreturn n+nfmake_pairp(nx o-> nupdatep(), nretp.nsecondp);
65 p}
66 npairo<nNodeo*p, nPairo> nget_segmentp(k+ktint nlp, k+ktint nrp){
67     nPair nret o= nsplitp(nrootp, nl o- l+m+mi1p);
68     kreturn n+nfmake_pairp(nretp.nfirstp, nsplitp(nretp.nsecondp, nr o- nl o+ l+m+mi1p));
69 p}

```

```

70 k+ktint nmainp(){
71     nullptr o= knew nNodep(nINFp, nINFp, l+m+mi0p);
72     nullptr o-> nchp[l+m+mi0p] o= nullptr o-> nchp[l+m+mi1p] o= nullptr;
73     nroot o= nullptr;
74 p}

```

Link/cut Tree

```

1 k+krinline k+ktvoid n+nfreversep(k+ktint nxp) p{
2     ntrp[nxp].nrev o+= l+m+mi1p; nswapp(ntrp[nxp].ncp[l+m+mi0p], ntrp[nxp].ncp[l+m+mi1p]);
3 p}
4
5 k+krinline k+ktvoid n+nfrotatep(k+ktint nxp, k+ktint nkp) p{
6     k+ktint ny o= ntrp[nxp].nfap, nz o= ntrp[nyp].nfap;
7     ntrp[nxp].nfa o= nzp; ntrp[nzp].ncp[ntrp[nzp].ncp[l+m+mi1p] o== nyp] o= nxp;
8     ntrp[ntrp[nxp].ncp[nk o^ l+m+mi1p]].nfa o= nyp; ntrp[nyp].ncp[nkp] o= ntrp[nxp].ncp[nk o^
    ↪ l+m+mi1p];
9     ntrp[nxp].ncp[nk o^ l+m+mi1p] o= nyp; ntrp[nyp].nfa o= nxp;
10 p}
11
12 k+krinline k+ktvoid n+nfsplayp(k+ktint nxp, k+ktint nwp) p{
13     k+ktint nz o= nxp; npushdownp(nxp);
14     kwhile p(ntrp[nxp].nfa o!= nwp) p{
15         k+ktint ny o= ntrp[nxp].nfap; nz o= ntrp[nyp].nfap;
16         kif p(nz o== nwp) p{
17             npushdownp(nz o= nyp); npushdownp(nxp);
18             nrotatep(nxp, ntrp[nyp].ncp[l+m+mi1p] o== nxp);
19             nupdatep(nyp); nupdatep(nxp);
20         p} kelse p{
21             npushdownp(nzp); npushdownp(nyp); npushdownp(nxp);
22             k+ktint nt1 o= ntrp[nyp].ncp[l+m+mi1p] o== nxp, nt2 o= ntrp[nzp].ncp[l+m+mi1p] o== nyp;
23             kif p(nt1 o== nt2p) nrotatep(nyp, nt2p), nrotatep(nxp, nt1p);
24             kelse nrotatep(nxp, nt1p), nrotatep(nxp, nt2p);
25             nupdatep(nzp); nupdatep(nyp); nupdatep(nxp);
26         p}
27     p}
28     nupdatep(nxp);
29     kif p(nx o!= nzp) nparp[nxp] o= nparp[nzp], nparp[nzp] o= l+m+mi0p;
30 p}
31
32 k+krinline k+ktvoid n+nfacessp(k+ktint nxp) p{
33     kfor p(k+ktint ny o= l+m+mi0p; nxp; ny o= nxp, nx o= nparp[nxp]) p{
34         nsplayp(nxp, l+m+mi0p);
35         kif p(ntrp[nxp].ncp[l+m+mi1p]) nparp[ntrp[nxp].ncp[l+m+mi1p]] o= nxp,
    ↪ ntrp[ntrp[nxp].ncp[l+m+mi1p]].nfa o= l+m+mi0p;
36         ntrp[nxp].ncp[l+m+mi1p] o= nyp; nparp[nyp] o= l+m+mi0p; ntrp[nyp].nfa o= nxp;
    ↪ nupdatep(nxp);
37     p}
38 p}
39
40 k+krinline k+ktvoid n+nfmakerootp(k+ktint nxp) p{
41     naccessp(nxp); nsplayp(nxp, l+m+mi0p); nreversep(nxp);
42 p}
43
44 k+krinline k+ktvoid n+nflinkp(k+ktint nxp, k+ktint nyp) p{
45     nmakerootp(nxp); nparp[nxp] o= nyp;

```

```

46 p}
47
48 k+krinline k+ktvoid n+nfcutp(k+ktint nxp, k+ktint nyp) p{
49     naccessp(nxp); nsplayp(nyp, l+m+mi0p);
50     kif p(nparp[nyp] o!= nxp) nswapp(nxp, nyp), naccessp(nxp), nsplayp(nyp, l+m+mi0p);
51     nparp[nyp] o= l+m+mi0p;
52 p}
53
54 k+krinline k+ktvoid n+nfsplitp(k+ktint nxp, k+ktint nyp) p{  c+c1// x will be the root of the
55     ↪ tree
56     nmakerootp(nyp); naccessp(nxp); nsplayp(nxp, l+m+mi0p);
57 p}

```

树状数组查询第 k 小元素

```

1 k+krinline k+ktvoid n+nfreversep(k+ktint nxp) p{
2     ntrp[nxp].nrev o^= l+m+mi1p; nswapp(ntrp[nxp].ncp[l+m+mi0p], ntrp[nxp].ncp[l+m+mi1p]);
3 p}
4
5 k+krinline k+ktvoid n+nfrotatep(k+ktint nxp, k+ktint nkp) p{
6     k+ktint ny o= ntrp[nxp].nfap, nz o= ntrp[nyp].nfap;
7     ntrp[nxp].nfa o= nzp; ntrp[nzp].ncp[ntrp[nzp].ncp[l+m+mi1p] o== nyp] o= nxp;
8     ntrp[ntrp[nxp].ncp[nk o^ l+m+mi1p]].nfa o= nyp; ntrp[nyp].ncp[nkp] o= ntrp[nxp].ncp[nk o^
9     ↪ l+m+mi1p];
10    ntrp[nxp].ncp[nk o^ l+m+mi1p] o= nyp; ntrp[nyp].nfa o= nxp;
11 p}
12
13 k+krinline k+ktvoid n+nfsplayp(k+ktint nxp, k+ktint nwp) p{
14     k+ktint nz o= nxp; npushdownp(nxp);
15     kwhile p(ntrp[nxp].nfa o!= nwp) p{
16         k+ktint ny o= ntrp[nxp].nfap; nz o= ntrp[nyp].nfap;
17         kif p(nz o== nwp) p{
18             npushdownp(nz o= nyp); npushdownp(nxp);
19             nrotatep(nxp, ntrp[nyp].ncp[l+m+mi1p] o== nxp);
20             nupdatep(nyp); nupdatep(nxp);
21         p} kelse p{
22             npushdownp(nzp); npushdownp(nyp); npushdownp(nxp);
23             k+ktint nt1 o= ntrp[nyp].ncp[l+m+mi1p] o== nxp, nt2 o= ntrp[nzp].ncp[l+m+mi1p] o== nyp;
24             kif p(nt1 o== nt2p) nrotatep(nyp, nt2p), nrotatep(nxp, nt1p);
25             kelse nrotatep(nxp, nt1p), nrotatep(nxp, nt2p);
26             nupdatep(nzp); nupdatep(nyp); nupdatep(nxp);
27         p}
28     p}
29     nupdatep(nxp);
30     kif p(nx o!= nzp) nparp[nxp] o= nparp[nzp], nparp[nzp] o= l+m+mi0p;
31 p}
32
33 k+krinline k+ktvoid n+nfaccessp(k+ktint nxp) p{
34     kfor p(k+ktint ny o= l+m+mi0p; nxp; ny o= nxp, nx o= nparp[nxp]) p{
35         nsplayp(nxp, l+m+mi0p);
36         kif p(ntrp[nxp].ncp[l+m+mi1p]) nparp[ntrp[nxp].ncp[l+m+mi1p]] o= nxp,
37         ↪ ntrp[ntrp[nxp].ncp[l+m+mi1p]].nfa o= l+m+mi0p;
38         ntrp[nxp].ncp[l+m+mi1p] o= nyp; nparp[nyp] o= l+m+mi0p; ntrp[nyp].nfa o= nxp;
39         ↪ nupdatep(nxp);
40     p}
41 p}

```

```

39 k+krinline k+ktvoid n+nfmakerootp(k+ktint nxp) p{
40     naccessp(nxp); nsplayp(nxp, l+m+mi0p); nreversep(nxp);
41 p}
42
43 k+krinline k+ktvoid n+nflinkp(k+ktint nxp, k+ktint nyp) p{
44     nmakerootp(nxp); nparp[nxp] o= nyp;
45 p}
46
47 k+krinline k+ktvoid n+nfcutp(k+ktint nxp, k+ktint nyp) p{
48     naccessp(nxp); nsplayp(nyp, l+m+mi0p);
49     kif p(nparp[nyp] o!= nxp) nswapp(nxp, nyp), naccessp(nxp), nsplayp(nyp, l+m+mi0p);
50     nparp[nyp] o= l+m+mi0p;
51 p}
52
53 k+krinline k+ktvoid n+nfsplitp(k+ktint nxp, k+ktint nyp) p{  c+c1// x will be the root of the
54     ↪ tree
55     nmakerootp(nyp); naccessp(nxp); nsplayp(nxp, l+m+mi0p);
56 p}

```

图论

基础

```

1 kstruct nGraph p{  c+c1// Remember to call .init()!
2     k+ktint nep, nnxtp[nMp], nvp[nMp], nadjp[nNp], nnp;
3     k+ktbool nbasep;
4     k+kr__inline k+ktvoid n+nfinitp(k+ktbool n_basep, k+ktint n_n o= l+m+mi0p) p{
5         nassertp(nn o< nNp);
6         nn o= n_np; nbase o= n_basep;
7         ne o= l+m+mi0p; nmemsetp(nadj o+ nbasep, o-l+m+mi1p, ksizeofp(o*nadjp) o* nnp);
8     p}
9     k+kr__inline k+ktint n+nfnew_nodep() p{
10         nadjp[nn o+ nbasep] o= o-l+m+mi1p;
11         nassertp(nn o+ nbase o+ l+m+mi1 o< nNp);
12         kreturn nno++ o+ nbasep;
13     p}
14     k+kr__inline k+ktvoid n+nfinisp(k+ktint nu0p, k+ktint nv0p) p{  c+c1// directional
15         nassertp(nu0 o< nn o+ nbase o&& nv0 o< nn o+ nbasedp);
16         nvp[nep] o= nv0p; nnxtp[nep] o= nadjp[nu0p]; nadjp[nu0p] o= neo++p;
17         nassertp(ne o< nMp);
18     p}
19     k+kr__inline k+ktvoid n+nfb_i_insp(k+ktint nu0p, k+ktint nv0p) p{  c+c1// bi-directional
20         ninisp(nu0p, nv0p); ninisp(nv0p, nu0p);
21     p}
22 p};

```

KM

```

1 kstruct nKM p{
2     c+c1// Truly O(n^3)
3     c+c1// 邻接矩阵，不能连的边设为 -INF，求最小权匹配时边权取负，但不能连的还是
4     ↪ -INF，使用时先对 1 -> n 调用 hungary()，再 get_ans() 求值
5     k+ktint nwp[nNp][nNp];
6     k+ktint nlxp[nNp], nlyp[nNp], nmatchp[nNp], nwayp[nNp], nslackp[nNp];

```



```

61     p}
62     kreturn nsump;
63 }
64 p} nkmp;

```

双连通分量

bcc.forest is a set of connected tree whose vertices are chequered with cut-vertex and BCC.

```

1 kconst k+ktbool nBCC_VERTEX o= l+m+mi0p, nBCC_EDGE o= l+m+mi1p;
2 kstruct nBCC p{ c+c1// N = N0 + M0. Remember to call init(&raw_graph).
3   nGraph o*ngp, nforestp; c+c1// g is raw graph ptr.
4   k+ktint ndfnp[nNp], nDFNp, nlowp[nNp];
5   k+ktint nstackp[nNp], ntopp;
6   k+ktint nexpan_top[nNp]; c+c1// Where edge i is expanded to in expaned graph.
7   c+c1// Vertex i expaned to i.
8   k+ktint ncompress_top[nNp]; c+c1// Where vertex i is compressed to.
9   k+ktbool nvertex_typep[nNp], ncutp[nNp], ncompress_cutp[nNp], nbranchp[nMp];
10  c+c1//std::vector<int> BCC_component[N]; // Cut vertex belongs to none.
11  k+kr__inline k+ktvoid n+nfnitp(nGraph o*nraw_graphp) p{
12    ng o= nraw_graphp;
13  p}
14  k+ktvoid n+nfnDFSp(k+ktint nup, k+ktint nep) p{
15    ndfnp[nup] o= nlowp[nup] o= o++nDFNp; ncutp[nup] o= n+nbfalsep;
16    kif p(o!~ngo->nadjp[nup]) p{
17      ncutp[nup] o= l+m+mi1p;
18      ncompress_top[nup] o= nforestp.nnew_nodep();
19      ncompress_cutp[ncompress_top[nup]] o= l+m+mi1p;
20    p}
21    kfor p(k+ktint ne o= ngo->nadjp[nup]; o~nep; ne o= ngo->nnxtp[nep]) p{
22      k+ktint nv o= ngo->nvp[nep];
23      kif p((ne o^ nep) o> l+m+mi1 o&& ndfnp[nvp] o> l+m+mi0 o&& ndfnp[nvp] o< ndfnp[nup])
→ p{
24        nstackp[ntopo++p] o= nep;
25        nlowp[nup] o= nstdo::nminp(nlowp[nup], ndfnp[nvp]);
26      p}
27      kelse kif p(o!ndfnp[nvp]) p{
28        nstackp[ntopo++p] o= nep; nbranchp[nep] o= l+m+mi1p;
29        nDFSp(nvp, nep);
30        nlowp[nup] o= nstdo::nminp(nlowp[nvp], nlowp[nup]);
31        kif p(nlowp[nvp] o>= ndfnp[nup]) p{
32          kif p(o!ncutp[nup]) p{
33            ncutp[nup] o= l+m+mi1p;
34            ncompress_top[nup] o= nforestp.nnew_nodep();
35            ncompress_cutp[ncompress_top[nup]] o= l+m+mi1p;
36          p}
37          k+ktint ncc o= nforestp.nnew_nodep();
38          nforestp.nbi_insp(ncompress_top[nup], nccp);
39          ncompress_cutp[nccp] o= l+m+mi0p;
40          c+c1//BCC_component[cc].clear();
41          kdo p{
42            k+ktint ncur_e o= nstackp[o--ntopp];
43            ncompress_top[nexpan_top[ncur_ep]] o= nccp;
44            ncompress_top[nexpan_top[ncur_eo^l+m+mi1p]] o= nccp;
45            kif p(nbranchp[ncur_ep]) p{
46              k+ktint nv o= ngo->nvp[ncur_ep];

```

```

47     kif p(ncutp[nvp])
48         nforestp.nbi_insp(nccp, ncompress_top[nvp]);
49     kelse p{
50         c+c1//BCC_component[cc].push_back(v);
51         ncompress_top[nvp] o= nccp;
52     p}
53 p}
54 p} kwhile p(nstackp[ntopp] o!= nep);
55 p}
56 p}
57 p}
58 p}
59 k+ktvoid n+nfsolvep() p{
60     nforestp.ninitp(ngo->nbasep);
61     k+ktint nn o= ngo->nnp;
62     kfor p(k+ktint ni o= l+m+mi0p; ni o< ngo->nep; nio++p) p{
63         nexpand_top[nip] o= ngo->nnew_nodep();
64     p}
65     nmemsetp(nbranchp, l+m+mi0p, ksizeofp(o*nbranchp) o* ngo->nep);
66     nmemsetp(ndfn o+ ngo->nbasep, l+m+mi0p, ksizeofp(o*ndfn) o* nnp); nDFN o= l+m+mi0p;
67     kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p)
68         kif p(o!ndfn[ni o+ ngo->nbasep]) p{
69             ntop o= l+m+mi0p;
70             nDFS[ni o+ ngo->nbasep, o-l+m+mi1p];
71         p}
72 p}
73 p} nbccp;
74
75 nbccp.ninitp(o&nraw_graphp);
76 nbccp.nsolvep();
77 c+c1// Do something with bcc.forest ...

```

边双连通分量

```

1 kstruct nBCC p{
2     nGraph o*ngp, nforestp;
3     k+ktint ndfn[nNp], nlowp[nNp], nstackp[nNp], ntotp[nNp], nbelongp[nNp], nvisp[nNp], ntopp,
4     ↪ ndfs_clockp;
5     c+c1// tot[] is the size of each BCC, belong[] is the BCC that each node belongs to
6     npairo<k+ktintp, k+ktint o> norip[nMp]; c+c1// bridge in raw_graph(raw node)
7     k+ktbool nis_bridgep[nMp];
8     k+kr__inline k+ktvoid n+nfinitp(nGraph o*nraw_graphp) p{
9         ng o= nraw_graphp;
10        nmemsetp(nis_bridgep, n+nbfalsep, ksizeofp(o*nis_bridgep) o* ng o-> nep);
11        nmemsetp(nvis o+ ng o-> nbasep, l+m+mi0p, ksizeofp(o*nvisp) o* ng o-> nnp);
12    p}
13    k+ktvoid n+nftarjanp(k+ktint nup, k+ktint nfromp) p{
14        ndfn[nup] o= nlowp[nup] o= o++ndfs_clockp; nvisp[nup] o= l+m+mi1p; nstackp[o++ntopp] o=
15        ↪ nup;
16        kfor p(k+ktint np o= ng o-> nadjp[nup]; o~npp; np o= ng o-> nnxtp[npp]) p{
17            kif p((np o^ l+m+mi1p) o== nfromp) kcontinuep;
18            k+ktint nv o= ng o-> nvp[npp];
19            kif p(nvisp[nvp]) p{
20                kif p(nvisp[nvp] o== l+m+mi1p) nlowp[nup] o= nminp(nlowp[nup], ndfn[nvp]);
21            p} kelse p{
22                ntarjanp(nvp, npp);

```

```

21        nlowp[nup] o= nminp(nlowp[nup], nlowp[nvp]);
22        kif p(nlowp[nvp] o> ndfn[nup]) nis_bridgep[np o/ l+m+mi2p] o= n+nbtruep;
23    p}
24 p}
25 kif p(ndfn[nup] o!= nlowp[nup]) kreturnp;
26 ntotp[nforestp.nnew_nodep()] o= l+m+mi0p;
27 kdo p{
28     nbelongp[nstackp[ntopp]] o= nforestp.nnp;
29     nvisp[nstackp[ntopp]] o= l+m+mi2p;
30     ntotp[nforestp.nnp]o++p;
31     o--ntopp;
32     p} kwhile p(nstackp[ntop o+ l+m+mi1p] o!= nup);
33 p}
34 k+ktvoid n+nfsolvep() p{
35     nforestp.ninitp(ng o-> nbasep);
36     k+ktint nn o= ng o-> nnp;
37     kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; o++nnp)
38         kif p(o!nvisp[ni o+ ng o-> nbasep]) p{
39             ntop o= ndfs_clock o= l+m+mi0p;
40             ntarjanp(ni o+ ng o-> nbasep, o-l+m+mi1p);
41         p}
42     kfor p(k+ktint ni o= l+m+mi0p; ni o< ng o-> ne o/ l+m+mi2p; o++nnp)
43         kif p(nis_bridgep[nip]) p{
44             k+ktint ne o= nforestp.nep;
45             nforestp.nbi_insp(nbelongp[ng o-> nvp[ni o* l+m+mi2p]], nbelongp[ng o-> nvp[ni o*
46             ↪ l+m+mi2 o+ l+m+mi1p]], ng o-> nwp[ni o* l+m+mi2p]);
47             norip[nep] o= nmake_pairp(ng o-> nvp[ni o* l+m+mi2 o+ l+m+mi1p], ng o-> nvp[ni o*
48             ↪ l+m+mi2p]);
49             norip[ne o+ l+m+mi1p] o= nmake_pairp(ng o-> nvp[ni o* l+m+mi2p], ng o-> nvp[ni o*
49             ↪ l+m+mi2 o+ l+m+mi1p]);
50         p}
51     p}
52 p} nbccp;

```

最小树形图

```

1 kconst k+ktint nMAXNp,nINFp;c+c1// INF >= sum( W_ij )
2 k+ktint nfromp[nMAXN o+ l+m+mi10p][nMAXN o* l+m+mi2 o+ l+m+mi10p],nnp,nmp,nedg[nMAXN o+
3     ↪ l+m+mi10p][nMAXN o* l+m+mi2 o+ l+m+mi10p];
4 k+ktint nselp[nMAXN o* l+m+mi2 o+ l+m+mi10p],nfap[nMAXN o* l+m+mi2 o+ l+m+mi10p],nvisp[nMAXN
5     ↪ o* l+m+mi2 o+ l+m+mi10p];
6 k+ktint n+nfgetfap(k+ktint nxp){kifp(nx o== nfap[nxp]) kreturn nxp; kreturn nfap[nxp] o=
7     ↪ ngetfap(nfap[nxp]);}
8 k+ktvoid n+nfliuzhup(){ c+c1// 1-base: root is 1, answer = (sel[i], i) for i in [2..n]
9     nfap[l+m+mi1p] o= l+m+mi1p;
10    kforp(k+ktint ni o= l+m+mi2p; ni o<= nnp; o++nnp){
11        nselp[nip] o= l+m+mi1p; nfap[nip] o= nip;
12        kforp(k+ktint nj o= l+m+mi1p; nj o<= nnp; o++nnp) kifp(nfap[njp] o!= nip)
13            kifp(nfromp[njp][nip] o= nip, nedg[nselp[nip]][nip] o> nedg[njp][nip]) nselp[nip]
14            ↪ o= njp;
15    p}
16    k+ktint nlimit o= nnp;
17    kwhilep(l+m+mi1p){
18        k+ktint nprelimit o= nlimitp; nmemsetp(nvisp, l+m+mi0p, ksizeofp(nvisp)); nvisp[l+m+mi1p]
19        ↪ o= l+m+mi1p;

```

```

15   kforp(k+ktint ni o= l+m+mi2p; ni o<= nprelimitp; o++nip) kifp(nfap[nip] o== ni o&&
    ↪ o!nvisp[nip]){
16   k+ktint nj o= nip; kwhilep(o!nvisp[njp]) nvisp[njp] o= nip, nj o= ngetfap(nselp[njp]);
17   kifp(nj o== l+m+mi1 o|| nvisp[njp] o!= nip) kcontinuep; nvectoro<k+ktinto> nCp; k+ktint
    ↪ nk o= njp;
18   kdo nCp.npush_backp(nkp), nk o= ngetfap(nselp[nkp]); kwhilep(nk o!= njp);
19   o++nlimitp;
20   kforp(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip){
21   nedgelp[nip][nlimitp] o= nINFp, nfromp[nip][nlimitp] o= nlimitp;
22   p}
23   nfap[nlimitp] o= nvisp[nlimitp] o= nlimitp;
24   kforp(k+ktint ni o= l+m+mi0p; ni o< k+ktintp(nCp.nsizep()); o++nip){
25   k+ktint nx o= nCp[nip], nfap[nxp] o= nlimitp;
26   kforp(k+ktint nj o= l+m+mi1p; nj o<= nnp; o++njp)
27   kifp(nedgelp[njp][nxp] o!= nINF o&& nedgelp[njp][nlimitp] o> nedgelp[njp][nxp] o-
    ↪ nedgelp[nselp[nxp]][nxp]){
28   nedgelp[njp][nlimitp] o= nedgelp[njp][nxp] o- nedgelp[nselp[nxp]][nxp];
29   nfromp[njp][nlimitp] o= nxp;
30   p}
31   p}
32   kforp(k+ktint njo=l+m+mi1p;njo<=nnp;o++njp) kifp(ngetfap(njp)o==nlimitp)
    ↪ nedgelp[njp][nlimitp] o= nINFp;
33   nselp[nlimitp] o= l+m+mi1p;
34   kforp(k+ktint nj o= l+m+mi1p; nj o<= nnp; o++njp)
35   kifp(nedgelp[nselp[nlimitp]][nlimitp] o> nedgelp[njp][nlimitp]) nselp[nlimitp] o= njp;
36   p}
37   kifp(nprelimit o== nlimitp) kbreakp;
38   p}
39   kforp(k+ktint ni o= nlimitp; ni o> l+m+mi1p; o--nip) nselp[nfromp[nselp[nip]][nip]] o=
    ↪ nselp[nip];
40   p}

```

带花树

```

1   nvectoro<k+ktinto> nlinkp[nmaxnp];
2   k+ktint nnp,nmatchp[nmaxnp],nQueueup[nmaxnp],nheadp,ntailp;
3   k+ktint npredp[nmaxnp],nbased[nmaxnp],nstartp,nfinishp,nnewbasep;
4   k+ktbool nInQueueup[nmaxnp],nInBlossomp[nmaxnp];
5   k+ktvoid n+nfpushp(k+ktint nup){ nQueueup[ntailo++p]o=nup;nInQueueup[nup]o=n+nbtruep; p}
6   k+ktint n+nfpoppp(){ kreturn nQueueup[nheado++p]; p}
7   k+ktint n+nfFindCommonAncestorp(k+ktint nup,k+ktint nvp){
8   k+ktbool nInPathp[nmaxnp];
9   kforp(k+ktint nio=l+m+mi0p;nio<nnp;nio++p) nInPathp[nip]o=l+m+mi0p;
10  kwhilep(n+nbtruep){ nuo=nbased[nup];nInPathp[nup]o=n+nbtruep;kifp(nuo==nstartp)
    ↪ kbreakp;nuo=npredp[nmatchp[nup]]; p}
11  kwhilep(n+nbtruep){ nvo=nbased[nvp];kifp(nInPathp[nvp]) kbreakp;nvo=npredp[nmatchp[nvp]];
    ↪ p}
12  kreturn nvp;
13  p}
14  k+ktvoid n+nfResetTracep(k+ktint nup){
15  k+ktint nvp;
16  kwhilep(nbased[nup]o!=nnewbasep){
17  nvo=nmatchp[nup];
18  nInBlossomp[nbased[nup]]o=nInBlossomp[nbased[nvp]]o=n+nbtruep;
19  nuo=npredp[nvp];
20  kifp(nbased[nup]o!=nnewbasep) npredp[nup]o=nvp;

```

```

21  p}
22  p}
23  k+ktvoid n+nfBlossompContractp(k+ktint nup,k+ktint nvp){
24  nnewbaseo=nFindCommonAncestorp(nup,nvp);
25  kfor p(k+ktint nio=l+m+mi0p;nio<nnp;nio++p)
26  nInBlossomp[nip]o=l+m+mi0p;
27  nResetTracep(nup);nResetTracep(nvp);
28  kifp(nbased[nup]o!=nnewbasep) npredp[nup]o=nvp;
29  kifp(nbased[nvp]o!=nnewbasep) npredp[nvp]o=nup;
30  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip)
31  kifp(nInBlossomp[nbased[nip]]){
32  nbased[nip]o=nnewbasep;
33  kifp(o!nInQueueup[nip]) npushp(nip);
34  p}
35  p}
36  k+ktbool n+nfFindAugmentingPathp(k+ktint nup){
37  k+ktbool nfoundo=n+nbfalsep;
38  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip) npredp[nip]o=-l+m+mi1p,nbased[nip]o=nip;
39  kfor p(k+ktint nio=l+m+mi0p;nio<nnp;nio++p) nInQueueup[nip]o=l+m+mi0p;
40  nstarto=nup;nfinisho=-l+m+mi1p; nheado=ntailo=l+m+mi0p; npushp(nstartp);
41  kwhilep(nheado<ntailp){
42  k+ktint nuo=npopp();
43  kforp(k+ktint nio=nlinkp[nup].nsizep()o-l+m+mi1p;nio>=l+m+mi0p;nio--p){
44  k+ktint nvo=nlinkp[nup][nip];
45  kifp(nbased[nup]o!=nbased[nvp]o&&nmatchp[nup]o!=nvp)
46  kifp(nvo==nstarto||p(nmatchp[nvp]o>=l+m+mi0o&&npredp[nmatchp[nvp]]o>=l+m+mi0p))
47  nBlossompContractp(nup,nvp);
48  kelse kifp(npredp[nvp]o==l+m+mi1p){
49  npredp[nvp]o=nup;
50  kifp(nmatchp[nvp]o>=l+m+mi0p) npushp(nmatchp[nvp]);
51  kelse{ nfinisho=nvp; kreturn n+nbtruep; p}
52  p}
53  p}
54  p}
55  kreturn nfoundp;
56  p}
57  k+ktvoid n+nfAugmentPathp(){
58  k+ktint nuo=nfinishp,nvp,nwp;
59  kwhilep(nuo>=l+m+mi0p){
    ↪ nvo=npredp[nup];nwo=nmatchp[nvp];nmatchp[nvp]o=nup;nmatchp[nup]o=nvp;nuo=nwp; p}
60  p}
61  k+ktvoid n+nfFindMaxMatchingsp(){
62  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip) nmatchp[nip]o=-l+m+mi1p;
63  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip) kifp(nmatchp[nip]o==l+m+mi1p)
    ↪ kifp(nFindAugmentingPathp(nip)) nAugmentPathp();
64  p}

```

Dominator Tree

```

1   nvectoro<k+ktinto> nprecip[nNp], nsuccp[nNp];
2   nvectoro<k+ktinto> nordp;
3   k+ktint nstamp, nvisp[nNp];
4   k+ktint nnump[nNp];
5   k+ktint nfap[nNp];
6   k+ktvoid n+nfdfsp(k+ktint nup) p{

```

```

7  nvisp[nup] = nstamp;
8  nnump[nup] = nordp.nsize();
9  nordp.npush_backp(nup);
10 kfor p(k+ktint ni = l+m+mi0p; ni < p(k+ktintp)nsuccp[nup].nsize(); o++nip) p{
11   k+ktint nv = nsuccp[nup][nip];
12   kif p(nvisp[nvp] o!= nstamp) p{
13     nfap[nvp] = nup;
14     ndfsp(nvp);
15   p}
16 p}
17 p}
18 k+ktint nfsp[nNp], nminsp[nNp], ndomp[nNp], nsemp[nNp];
19 k+ktint n+nfFindp(k+ktint nup) p{
20   kif p(nu o!= nfsp[nup]) p{
21     k+ktint nv = nfsp[nup];
22     nfsp[nup] = nfindp(nfsp[nup]);
23     kif p(nminsp[nvp] o!= o-l+m+mi1 o&& nnump[nsemp[nminsp[nvp]]] o<
    ↪ nnump[nsemp[nminsp[nup]]]) p{
24       nminsp[nup] = nminsp[nvp];
25     p}
26 p}
27 kreturn nfsp[nup];
28 p}
29 k+ktvoid n+nfmergcp(k+ktint nup, k+ktint nvp) p{ nfsp[nup] = nvp; p}
30 nvectoro<k+ktinto> nbufp[nNp];
31 k+ktint nbuf2p[nNp];
32 k+ktvoid n+nfmarkp(k+ktint nsourcep) p{
33   nordp.nclearp();
34   o++nstamp;
35   ndfsp(nsourcep);
36   kfor p(k+ktint ni = l+m+mi0p; ni < p(k+ktintp)nordp.nsize(); o++nip) p{
37     k+ktint nu = nordp[nip];
38     nfsp[nup] = nup, nminsp[nup] = o-l+m+mi1p, nbuf2p[nup] = o-l+m+mi1p;
39   p}
40 kfor p(k+ktint ni = p(k+ktintp)nordp.nsize() o- l+m+mi1p; ni > l+m+mi0p; o--nip) p{
41   k+ktint nu = nordp[nip], np = nfap[nup];
42   nsemp[nup] = npp;
43   kfor p(k+ktint nj = l+m+mi0p; nj < p(k+ktintp)nprecip[nup].nsize(); o++njp) p{
44     k+ktint nv = nprecip[nup][njp];
45     kif p(nusep[nvp] o!= nstamp) kcontinuep;
46     kif p(nnump[nvp] > nnump[nup]) p{
47       nfindp(nvp); nv = nsemp[nminsp[nvp]];
48     p}
49     kif p(nnump[nvp] < nnump[nsemp[nup]]) p{
50       nsemp[nup] = nvp;
51     p}
52 p}
53 nbufp[nsemp[nup]].npush_backp(nup);
54 nminsp[nup] = nup;
55 nmergcp(nup, npp);
56 kwhile p(nbufp[npp].nsize()) p{
57   k+ktint nv = nbufp[npp].nbackp();
58   nbufp[npp].npop_backp();
59   nfindp(nvp);
60   kif p(nsemp[nvp] o== nsemp[nminsp[nvp]]) p{

```

```

61   ndomp[nvp] = nsemp[nvp];
62   p} kelse p{
63     nbuf2p[nvp] = nminsp[nvp];
64   p}
65 p}
66 p}
67 ndomp[nordp[l+m+mi0p]] = nordp[l+m+mi0p];
68 kfor p(k+ktint ni = l+m+mi0p; ni < p(k+ktintp)nordp.nsize(); o++nip) p{
69   k+ktint nu = nordp[nip];
70   kif p(o~nbuf2p[nup]) p{
71     ndomp[nup] = ndomp[nbuf2p[nup]];
72   p}
73 p}
74 p}

```

无向图最小割

```

1 k+ktint ncostp[nmaxnp][nmaxnp], nseqp[nmaxnp], nlenp[nmaxnp], nnp, nmp, npopp, nansp;
2 k+ktbool nusedp[nmaxnp];
3 k+ktvoid n+nfInitp(){
4   k+ktint nip, njp, nap, nbp, ncp;
5   kforp(nio=l+m+mi0p; nio<nnp; nio++p) kforp(njo=l+m+mi0p; njo<nnp; njo++p)
    ↪ ncostp[nip][njp]o=l+m+mi0p;
6   kforp(nio=l+m+mi0p; nio<nmp; nio++p){
7     nscanfp(l+s"%d %d %d"p, o&nap, o&nbp, o&ncp); ncostp[nap][nbp]o=ncp;
    ↪ ncostp[nbp][nap]o=ncp;
8   p}
9   npopo=nnp; kforp(nio=l+m+mi0p; nio<nnp; nio++p) nseqp[nip]o=nip;
10 p}
11 k+ktvoid n+nfWorkp(){
12   nanso=ninfp; k+ktint nip, njp, nkp, nlp, nmmp, nsump, npkp;
13   kwhilep(npop > l+m+mi1p){
14     kforp(nio=l+m+mi1p; nio<npop; nio++p) nusedp[nseqp[nip]]o=l+m+mi0p;
    ↪ nusedp[nseqp[l+m+mi0p]]o=l+m+mi1p;
15     kforp(nio=l+m+mi1p; nio<npop; nio++p)
    ↪ nlenp[nseqp[nip]]o=ncostp[nseqp[l+m+mi0p]][nseqp[nip]];
16     npko=l+m+mi0p; nmmpo=-ninfp; nko=-l+m+mi1p;
17     kforp(nio=l+m+mi1p; nio<npop; nio++p) kifp(nlenp[nseqp[nip]] > nmmp){
    ↪ nmmpo=nlenp[nseqp[nip]]; nko=nip; p}
18     kforp(nio=l+m+mi1p; nio<npop; nio++p){
19       nusedp[nseqp[nlo=nkp]]o=l+m+mi1p;
20       kifp(nio==npopo-l+m+mi2p) npko=nkp;
21       kifp(nio==npopo-l+m+mi1p) kbreakp;
22       nmmpo=-ninfp;
23       kforp(njo=l+m+mi1p; njo<npop; njo++p) kifp(o!nusedp[nseqp[njp]])
24         kifp((nlenp[nseqp[njp]]o+ncostp[nseqp[nlp]][nseqp[njp]]) > nmmp)
25         nmmpo=nlenp[nseqp[njp]], nko=njp;
26     p}
27     nsumo=l+m+mi0p;
28     kforp(nio=l+m+mi0p; nio<npop; nio++p) kifp(ni o!= nkp)
    ↪ nsumo+ncostp[nseqp[nkp]][nseqp[nip]];
29     nanso=nminp(nansp, nsump);
30     kforp(nio=l+m+mi0p; nio<npop; nio++p)
31       ncostp[nseqp[nkp]][nseqp[nip]]o=ncostp[nseqp[nip]][nseqp[nkp]]o+ncostp[nseqp[nkp]][nseqp[nip]];
32     nseqp[nkp]o=nseqp[o--npop];
33   p}

```

```

34 nprintf(l+s"%dl+s+se\nl+s"p,nansp);
35 p}

```

重口味费用流

```

1 k+ktint nSp, nTp, ntotFlowp, ntotCostp;
2
3 k+ktint ndisp[nNp], nslackp[nNp], nvisitp[nNp];
4
5 k+ktint n+nfmactable p() p{
6     k+ktint ndelta o= nINFp;
7     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p) p{
8         kif p(o!nvisitp[nip] o&& nslackp[nip] o< ndeltap) ndelta o= nslackp[nip];
9         nslackp[nip] o= nINFp;
10    p}
11    kif p(ndelta o== nINFp) kreturn l+m+mi1p;
12    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p)
13        kif p(nvisitp[nip]) ndisp[nip] o+= ndeltap;
14    kreturn l+m+mi0p;
15 p}
16
17 k+ktint n+nfdfs p(k+ktint nxp, k+ktint nflowp) p{
18     kif p(nx o== nTp) p{
19         ntotFlow o+= nflowp;
20         ntotCost o+= nflow o* p(ndisp[nSp] o- ndisp[nTp]);
21         kreturn nflowp;
22     p}
23     nvisitp[nxp] o= l+m+mi1p;
24     k+ktint nleft o= nflowp;
25     kfor p(k+ktint ni o= nep.nlastp[nxp]; o~nip; ni o= nep.nsuccp[nip])
26         kif p(nep.ncapp[nip] o> l+m+mi0 o&& o!nvisitp[nep.notherp[nip]]) p{
27             k+ktint ny o= nep.notherp[nip];
28             kif p(ndisp[nyp] o+ nep.ncostp[nip] o== ndisp[nxp]) p{
29                 k+ktint ndelta o= ndfs p(nyp, nmin p(nleftp, nep.ncapp[nip]));
30                 nep.ncapp[nip] o-= ndeltap;
31                 nep.ncapp[ni o^ l+m+mi1p] o+= ndeltap;
32                 nleft o-= ndeltap;
33                 kif p(o!nleftp) p{ nvisitp[nxp] o= l+m+mi0p; kreturn nflowp; p}
34             p} kelse p{
35                 nslackp[nyp] o= nmin p(nslackp[nyp], ndisp[nyp] o+ nep.ncostp[nip] o-
36                 ↪ ndisp[nxp]);
37             p}
38             kreturn nflow o- nleftp;
39 p}
40
41 npair o<k+ktintp, k+ktinto> nminCost p() p{
42     ntotFlow o= l+m+mi0p; ntotCost o= l+m+mi0p;
43     nfill p(ndis o+ l+m+mi1p, ndis o+ nT o+ l+m+mi1p, l+m+mi0p);
44     kdo p{
45         kdo p{
46             nfill p(nvisit o+ l+m+mi1p, nvisit o+ nT o+ l+m+mi1p, l+m+mi0p);
47         p} kwhile p(ndfs p(nSp, nINFp));
48     p} kwhile p(o!nmodlable p());
49     kreturn n+nfmake_pair p(ntotFlowp, ntotCostp);

```

```

50 p}

```

2-SAT

```

1 k+ktint nSp, nTp, ntotFlowp, ntotCostp;
2
3 k+ktint ndisp[nNp], nslackp[nNp], nvisitp[nNp];
4
5 k+ktint n+nfmactable p() p{
6     k+ktint ndelta o= nINFp;
7     kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p) p{
8         kif p(o!nvisitp[nip] o&& nslackp[nip] o< ndeltap) ndelta o= nslackp[nip];
9         nslackp[nip] o= nINFp;
10    p}
11    kif p(ndelta o== nINFp) kreturn l+m+mi1p;
12    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p)
13        kif p(nvisitp[nip]) ndisp[nip] o+= ndeltap;
14    kreturn l+m+mi0p;
15 p}
16
17 k+ktint n+nfdfs p(k+ktint nxp, k+ktint nflowp) p{
18     kif p(nx o== nTp) p{
19         ntotFlow o+= nflowp;
20         ntotCost o+= nflow o* p(ndisp[nSp] o- ndisp[nTp]);
21         kreturn nflowp;
22     p}
23     nvisitp[nxp] o= l+m+mi1p;
24     k+ktint nleft o= nflowp;
25     kfor p(k+ktint ni o= nep.nlastp[nxp]; o~nip; ni o= nep.nsuccp[nip])
26         kif p(nep.ncapp[nip] o> l+m+mi0 o&& o!nvisitp[nep.notherp[nip]]) p{
27             k+ktint ny o= nep.notherp[nip];
28             kif p(ndisp[nyp] o+ nep.ncostp[nip] o== ndisp[nxp]) p{
29                 k+ktint ndelta o= ndfs p(nyp, nmin p(nleftp, nep.ncapp[nip]));
30                 nep.ncapp[nip] o-= ndeltap;
31                 nep.ncapp[ni o^ l+m+mi1p] o+= ndeltap;
32                 nleft o-= ndeltap;
33                 kif p(o!nleftp) p{ nvisitp[nxp] o= l+m+mi0p; kreturn nflowp; p}
34             p} kelse p{
35                 nslackp[nyp] o= nmin p(nslackp[nyp], ndisp[nyp] o+ nep.ncostp[nip] o-
36                 ↪ ndisp[nxp]);
37             p}
38             kreturn nflow o- nleftp;
39 p}
40
41 npair o<k+ktintp, k+ktinto> nminCost p() p{
42     ntotFlow o= l+m+mi0p; ntotCost o= l+m+mi0p;
43     nfill p(ndis o+ l+m+mi1p, ndis o+ nT o+ l+m+mi1p, l+m+mi0p);
44     kdo p{
45         kdo p{
46             nfill p(nvisit o+ l+m+mi1p, nvisit o+ nT o+ l+m+mi1p, l+m+mi0p);
47         p} kwhile p(ndfs p(nSp, nINFp));
48     p} kwhile p(o!nmodlable p());
49     kreturn n+nfmake_pair p(ntotFlowp, ntotCostp);
50 p}

```

其他

Dancing Links

```

1 kstruct nNode p{
2     nNode o*nlp, o*nrp, o*nup, o*ndp, o*ncolp;
3     k+ktint nsizep, nline_nop;
4     nNodep() p{
5         nsize o= l+m+mi0p; nline_no o= o-l+m+mi1p;
6         nl o= nr o= nu o= nd o= ncol o= n+nbNULLp;
7     p}
8 p} o*nrootp;
9
10 k+ktvoid n+nfcoverp(nNode o*ncp) p{
11     nco->nlo->nr o= nco->nrp; nco->nro->nl o= nco->nlp;
12     kfor p(nNode o*nu o= nco->ndp; nu o!= ncp; nu o= nuo->ndp)
13         kfor p(nNode o*nv o= nuo->nrp; nv o!= nup; nv o= nvo->nrp) p{
14             nvo->ndo->nu o= nvo->nup;
15             nvo->nuo->nd o= nvo->ndp;
16             o-- nvo->ncolo->nsizep;
17         p}
18 p}
19
20 k+ktvoid n+nfcoverp(nNode o*ncp) p{
21     kfor p(nNode o*nu o= nco->nup; nu o!= ncp; nu o= nuo->nup) p{
22         kfor p(nNode o*nv o= nuo->nlp; nv o!= nup; nv o= nvo->nlp) p{
23             o++ nvo->ncolo->nsizep;
24             nvo->nuo->nd o= nvp;
25             nvo->ndo->nu o= nvp;
26         p}
27     p}
28     nco->nlo->nr o= ncp; nco->nro->nl o= ncp;
29 p}
30
31 nstdo::nvectoro<k+ktinto> nanswerp;
32 k+ktbool n+nfsearchp(k+ktint nkp) p{
33     kif p(nrooto->nr o== nrootp) kreturn n+nbtruep;
34     nNode o*nrr o= n+nbNULLp;
35     kfor p(nNode o*nu o= nrooto->nrr; nu o!= nrootp; nu o= nuo->nrr)
36         kif p(nr o== n+nbNULL o|| nuo->nsize o< nro->nsizep)
37             nr o= nup;
38     kif p(nr o== n+nbNULL o|| nro->nsize o== l+m+mi0p) kreturn n+nbfalsep;
39     kelse p{
40         ncoverp(nrr);
41         k+ktbool nsucc o= n+nbfalsep;
42         kfor p(nNode o*nu o= nro->ndp; nu o!= nr o&& o!nsuccp; nu o= nuo->ndp) p{
43             nanswerp.npush_backp(nuo->nline_nop);
44             kfor p(nNode o*nv o= nuo->nrr; nv o!= nup; nv o= nvo->nrr) c+c1// Cover row
45                 ncoverp(nvo->ncolp);
46             nsucc o|= nsearchp(nk o+ l+m+mi1p);
47             kfor p(nNode o*nv o= nuo->nlp; nv o!= nup; nv o= nvo->nlp)
48                 nuncoverp(nvo->ncolp);
49             kif p(o!nsuccp) nanswerp.npop_backp();
50         p}
51     nuncoverp(nrr);

```

```

52     kreturn nsuccp;
53 p}
54 p}
55
56 k+ktbool nentryp[nCRp][nCCp];
57 nNode o*nwhop[nCRp][nCCp];
58 k+ktint ncrp, nccp;
59
60 k+ktvoid n+nfconstructp() p{
61     nroot o= knew nNodep();
62     nNode o*nlast o= nrootp;
63     kfor p(k+ktint ni o= l+m+mi0p; ni o< nccp; o++ nip) p{
64         nNode o*nu o= knew nNodep();
65         nlasto->nr o= nup; nuo->nl o= nlastp;
66         nNode o*nv o= nup; nuo->nline_no o= nip;
67         nlast o= nup;
68         kfor p(k+ktint nj o= l+m+mi0p; nj o< ncrp; o++ njp)
69             kif p(nentryp[njp][nip]) p{
70                 o++ nuo->nsizep;
71                 nNode o*ncur o= knew nNodep();
72                 nwhop[njp][nip] o= ncurp;
73                 ncuro->nline_no o= njp;
74                 ncuro->ncol o= nup;
75                 ncuro->nu o= nvp; nvo->nd o= ncurp;
76                 nv o= ncurp;
77             p}
78         nvo->nd o= nup; nuo->nu o= nvp;
79     p}
80     nlasto->nr o= nrootp; nrooto->nl o= nlastp;
81     kfor p(k+ktint nj o= l+m+mi0p; nj o< ncrp; o++ njp) p{
82         nNode o*nlast o= n+nbNULLp;
83         kfor p(k+ktint ni o= ncc o- l+m+mi1p; ni o>= l+m+mi0p; o-- nip)
84             kif p(nentryp[njp][nip]) p{
85                 nlast o= nwhop[njp][nip];
86                 kbreakp;
87             p}
88         kfor p(k+ktint ni o= l+m+mi0p; ni o< nccp; o++ nip)
89             kif p(nentryp[njp][nip]) p{
90                 nlasto->nr o= nwhop[njp][nip];
91                 nwhop[njp][nip]o->nl o= nlastp;
92                 nlast o= nwhop[njp][nip];
93             p}
94     p}
95 p}
96
97 k+ktvoid n+nfdestructp() p{
98     kfor p(nNode o*nu o= nrooto->nrr; nu o!= nrootp; p) p{
99         kfor p(nNode o*nv o= nuo->ndp; nv o!= nup; p) p{
100             nNode o*nnxt o= nvo->ndp;
101             kdeletep(nvp);
102             nv o= nnxtp;
103         p}
104         nNode o*nnxt o= nuo->nrr;
105         kdeletep(nup); nu o= nnxtp;
106     p}

```

```
107 kdelete nrootp;
108 p}
```

蔡勒公式
0 for Sunday. Day and month is 1-based.

```
1 k+ktint n+nfzellerp(k+ktint nyp,k+ktint nmp,k+ktint ndp) p{
2   kif p(nmo<=l+m+mi2p) nyo=-p,nmo+=l+m+mi12p; k+ktint nco=nyo/l+m+mi100p; nyo%=l+m+mi100p;
3   k+ktint nwo=p((nco>>l+m+mi2p)o-
    ↪ p(nco<<l+m+mi1p)o+nyo+p(nyo>>l+m+mi2p)o+p(l+m+mi13o*p(nmo+l+m+mi1p)o/l+m+mi5p)o+ndo-
    ↪ l+m+mi1p)o%l+m+mi7p;
4   kif p(nwo<l+m+mi0p) nwo+=l+m+mi7p; kreturnp(nwp);
5   p}
```

技巧

真正的释放 STL 容器内存空间

```
1 ktemplate o<typename nTo>
2 k+kr__inline k+ktvoid nclearp(nTo& ncontainerp) p{
3   ncontainerp.nclearp(); c+c1// 或者删除了一堆元素
4   nTp(ncontainerp).nswapp(ncontainerp);
5   p}
```

无敌的大整数相乘取模
Time complexity $O(1)$.

```
1 c+c1// 需要保证 x 和 y 非负
2 k+ktlong k+ktlong n+nfmultp(k+ktlong k+ktlong nxp, k+ktlong k+ktlong nyp, k+ktlong k+ktlong
    ↪ nMODNp) p{
3   k+ktlong k+ktlong nt o= p(nx o* ny o- p(k+ktlong k+ktlongp)((k+ktlong k+ktdoublep)nx o/
    ↪ nMODN o* ny o+ l+m+mf1e-3p) o* nMODNp) o% nMODNp;
4   kreturn nt o< l+m+mi0 o? nt o+ n+nMODN p: ntp;
5   p}
```

无敌的读入优化

```
1 c+c1// getchar() 读入优化 << 关同步 cin << 此优化
2 c+c1// 用 isdigit() 会小幅变慢
3 c+c1// 返回 false 表示读到文件尾
4 knamespace nReader p{
5   kconst k+ktint nL o= p(l+m+mi1 o<< l+m+mi15p) o+ l+m+mi5p;
6   k+ktchar nbufferp[nLp], o*nSp, o*nTp;
7   k+kr__inline k+ktbool n+nfgetcharp(k+ktchar o&nchp) p{
8     kif p(nS o== nTp) p{
9       nT o= p(nS o= nbufferp) o+ nfreaddp(nbufferp, l+m+mi1p, nLp, nstdinp);
10      kif p(nS o== nTp) p{
11        nch o= nEOFp;
12        kreturn n+nbfalsep;
13      p}
14    p}
15    nch o= o*nSo++p;
16    kreturn n+nbtruep;
17    p}
18 k+kr__inline k+ktbool n+nfgetintp(k+ktint o&nxp) p{
```

```
19 k+ktchar nchp; k+ktbool nneg o= l+m+mi0p;
20 kfor p(; ngetcharp(nchp) o&& p(nch o< l+s+sc'0' o|| nch o> l+s+sc'9'p); p) nneg o^= nch
    ↪ o== l+s+sc'-'p;
21 kif p(nch o== nEOFp) kreturn n+nbfalsep;
22 nx o= nch o- l+s+sc'0'p;
23 kfor p(; ngetcharp(nchp), nch o>= l+s+sc'0' o&& nch o<= l+s+sc'9'p; p)
    ↪ nx o= nx o* l+m+mi10 o+ nch o- l+s+sc'0'p;
24 kif p(nnegp) nx o= o-nxp;
25 kreturn n+nbtruep;
26 p}
27 p}
28 p}
```

梅森旋转算法

High quality pseudorandom number generator, twice as efficient as rand() with -02. C++11 required.

```
1 c+cp#include c+cpf<random>
2
3 k+ktint n+nfmainp() p{
4   nstdo::nmt19937 ngp(nseedp); c+c1// std::mt19937_64
5   nstdo::ncout o<< ngp() o<< nstdo::nendlp;
6   p}
```

提示

控制 cout 输出实数精度

```
1 nstdo::ncout o<< nstdo::nfixed o<< nstdo::nsetprecisionp(l+m+mi5p);
```

vimrc

```
1 set nu
2 set sw=4
3 set sts=4
4 set ts=4
5 syntax on
6 set cindent
```

让 make 支持 c ++ 11

In .bashrc or whatever:

```
export CXXFLAGS='-std=c++11 -Wall'
```

tuple 相关

```
1 nmytuple o= nstdo::nmake_tuple p(l+m+mi10p, l+m+mf2.6p, l+s+sc'a'p); c+c1// packing
    ↪ values into tuple
2 nstdo::ntie p(nmyintp, nstdo::nignorep, nmycharp) o= nmytuplep; c+c1// unpacking tuple into
    ↪ variables
3 nstdo::ngeto<nIo>p(nmytuplep) o= l+m+mi20p;
4 nstdo::ncout o<< nstdo::ngeto<nIo>p(nmytuplep) o<< nstdo::nendlp; c+c1// get the
    ↪ lth(const) element
```

线性规划转对偶

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & \mathbf{A} \mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0 \end{array} \iff \begin{array}{ll} \text{minimize} & \mathbf{y}^T \mathbf{b} \\ \text{subject to} & \mathbf{y}^T \mathbf{A} \geq \mathbf{c}^T, \mathbf{y} \geq 0 \end{array}$$

32-bit/64-bit 随机素数

32-bit	64-bit
73550053	1249292846855685773
148898719	1701750434419805569
189560747	3605499878424114901
459874703	5648316673387803781
1202316001	6125342570814357977
1431183547	6215155308775851301
1438011109	6294606778040623451
1538762023	6347330550446020547
1557944263	7429632924303725207
1981315913	8524720079480389849

NTT 素数及其原根

Prime	Primitive root
1053818881	7
1051721729	6
1045430273	3
1012924417	5
1007681537	3

Java Hints

```

1 k+knimport n+nnjava.io.*o;
2 k+knimport n+nnjava.util.*o;
3 k+knimport n+nnjava.math.*o;
4
5 k+kdpublic k+kdclass n+ncMain o{
6     k+kdstatic k+ktint n+nfgeto(k+ktchar nco) o{
7         kif o(nc o<= l+s+sc'9'o)
8             kreturn nc o- l+s+sc'0'o;
9         kelse kif o(nc o<= l+s+sc'Z'o)
10            kreturn nc o- l+s+sc'A' o+ l+m+mi10o;
11        kelse
12            kreturn nc o- l+s+sc'a' o+ l+m+mi36o;
13    o}
14 k+kdstatic k+ktchar n+nfgeto(k+ktint nxo) o{
15     kif o(nx o<= l+m+mi9o)
16         kreturn o(k+ktcharo)(nx o+ l+s+sc'0'o);
17     kelse kif o(nx o<= l+m+mi35o)
18         kreturn o(k+ktcharo)(nx o- l+m+mi10 o+ l+s+sc'A'o);
19     kelse
20         kreturn o(k+ktcharo)(nx o- l+m+mi36 o+ l+s+sc'a'o);
21 o}
22 k+kdstatic nBigInteger n+nfgeto(nString nso, nBigInteger nxo) o{
23     nBigInteger nans o= nBigIntegero.n+navalueOfo(l+m+mi0o), nnow o=
24     ↪ nBigIntegero.n+navalueOfo(l+m+mi1o);
25     kfor o(k+ktint ni o= nso.n+nalengtho() o- l+m+mi1o; ni o>= l+m+mi0o; nio--) o{
26         nans o=
27         ↪ nanso.n+naaddo(nnowo.n+namultiplyo(nBigIntegero.n+navalueOfo(ngeto(nso.n+nacharAto(nio)))));
28         nnow o= nnowo.n+namultiplyo(nxo);
29     o}
30     kreturn nanso;

```

```

29 o}
30 k+kdpublic k+kdstatic k+ktvoid n+nfmaino(nString o[] nargso) o{
31     nScanner ncin o= knew nScannero(knew nBufferedInputStreamo(nSystemo.n+naino));
32     kfor o(; o; o) o{
33         nBigInteger nx o= ncino.n+nanextBigIntegero();
34         kif o(nxo.n+nacompareToo(nBigIntegero.n+navalueOfo(l+m+mi0o)) o== l+m+mi0o)
35             kbreako;
36         nString ns o= ncino.n+nanexto(), nt o= ncino.n+nanexto(), nr o= l+s""o;
37         nBigInteger nans o= ngeto(nso, nxo).n+namodo(ngeto(nts, nxo));
38         kif o(nanso.n+nacompareToo(nBigIntegero.n+navalueOfo(l+m+mi0o)) o== l+m+mi0o)
39             nr o= l+s"0"o;
40         kfor o(; nanso.n+nacompareToo(nBigIntegero.n+navalueOfo(l+m+mi0o)) o> l+m+mi0o;) o{
41             nr o= ngeto(nanso.n+namodo(nxo).n+naintValueo()) o+ nro;
42             nans o= nanso.n+nadivideo(nxo);
43         o}
44         nSystemo.n+naouto.n+naprintlno(nr);
45     o}
46 o}
47
48 c+c1// Arrays
49 k+ktint nao[];
50 o.n+nafill(nao, k+ktint nfromIndexo, k+ktint ntoIndexo, nvalo); o| o.n+nasorto(nao,
51     ↪ k+ktint nfromIndexo, k+ktint ntoIndexo])
52 c+c1// String
53 nString nso;
54 o.n+nacharAto(k+ktint nio); o| ncompareToo(nStringo) o| ncompareToIgnoreCase o() o|
55     ↪ ncontainso(nStringo) o|
56 nlength o() o| nsubstringo(k+ktint nlo, k+ktint nleno)
57 c+c1// BigInteger
58 o.n+naabso() o| o.n+naaddo() o| nbitLength o() o| nsubtract o() o| ndivide o() o| nremainder
59     ↪ o() o| ndivideAndRemainder o() o| nmodPow(nbo, nco) o|
60 npowo(k+ktinto) o| nmultiply o() o| ncompareTo o() o|
61 ngcd() o| nintValue o() o| nlongValue o() o| nisProbablePrimeo(k+ktint nco) o(l+m+mi1 o-
62     ↪ l+m+mi1o/l+m+mi2o^nc) o|
63 nnextProbablePrime o() o| nshiftLefto(k+ktinto) o| nvalueOf o()
64 c+c1// BigDecimal
65 o.n+naROUND_CEILING o| nROUND_DOWN_FLOOR o| nROUND_HALF_DOWN o| nROUND_HALF_EVEN o|
66     ↪ nROUND_HALF_UP o| nROUND_UP
67 o.n+nadivideo(nBigDecimal nbo, k+ktint nscale o, k+ktint nround_modeo) o| ndoubleValue o() o|
68     ↪ nmovePointLefto(k+ktinto) o| npowo(k+ktinto) o|
69 nsetScaleo(k+ktint nscale o, k+ktint nround_modeo) o| nstripTrailingZeros o()
70 c+c1// StringBuilder
71 nStringBuilder nsb o= knew nStringBuilder o();
72 nsbo.n+naappendo(nelemo) o| nouto.n+naprintlno(nsb);

```