Arondight's Standard Code Library*

Shanghai Jiao Tong University

Dated: August 11, 2017

*https://www.github.com/footoredo/Arondight

计算几何

```
k+ktint n+nfsignp(nDB nxp) p{
  kreturn p(nx o> nepsp) o- p(nx o< o-nepsp);
p}
nDB n+nfmsqrtp(nDB nxp) p{
  kreturn nsignp(nxp) o> l+m+mi0 o? nsqrtp(nxp) o: l+m+mi0p;
p}

kstruct nPoint p{
  nDB nxp, nyp;
  nPoint n+nfrotatep(nDB nangp) kconst p{  c+c1// 逆时针旋转 ang 弧度
    kreturn nPointp(ncosp(nangp) o* nx o- nsinp(nangp) o* nyp,
        ncosp(nangp) o* ny o+ nsinp(nangp) o* nxp);
  p}
  nPoint n+nfturn90p() kconst p{  c+c1// 逆时针旋转 90 度
    kreturn nPointp(o-nyp, nxp);
  p}
  nPoint n+nfunitp() kconst p{
    kreturn o*kthis o/ nlenp();
  p}
p};
nDB n+nfdotp(kconst nPointo& nap, kconst nPointo& nbp) p{
  kreturn nap.nx o* nbp.nx o+ nap.ny o* nbp.nyp;
p}
nDB n+nfdetp(kconst nPointo& nap, kconst nPointo& nbp) p{
  kreturn nap.nx o* nbp.ny o- nap.ny o* nbp.nxp;
p}
c+cp#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
c+cp#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
k+ktbool n+nfisLLp(kconst nLineo& nl1p, kconst nLineo& nl2p, nPointo& npp) p{  c+c1// 直线
  ↪ 与直线交点
  nDB ns1 o= ndetp(nl2p.nb o- nl2p.nap, nl1p.na o- nl2p.nap),
     ns2 o= o-ndetp(nl2p.nb o- nl2p.nap, nl1p.nb o- nl2p.nap);
  kif p(o!nsignp(ns1 o+ ns2p)) kreturn n+nbfalsep;
  np o= p(nl1p.na o* ns2 o+ nl1p.nb o* ns1p) o/ p(ns1 o+ ns2p);
  kreturn n+nbtruep;
p}
k+ktbool n+nfonSegp(kconst nLineo& nlp, kconst nPointo& npp) p{  c+c1// 点在线段上
  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;
p}
nPoint n+nfprojectionp(kconst nLine o& nlp, kconst nPointo& npp) p{
  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());
p}
nDB n+nfdisToLinep(kconst nLineo& nlp, kconst nPointo& npp) p{  c+c1// 点到 * 直线 * 距离
  kreturn nfabsp(ndetp(np o- nlp.nap, nlp.nb o- nlp.nap) o/ p(nlp.nb o- nlp.nap).nlenp());
p}
nDB n+nfdisToSegp(kconst nLineo& nlp, kconst nPointo& npp) p{  c+c1// 点到线段距离
  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());
p}
c+c1// 圆与直线交点
k+ktbool n+nfisCLp(nCircle nap, nLine nlp, nPointo& np1p, nPointo& np2p) p{
  nDB nx o= ndotp(nlp.na o- nap.nop, nlp.nb o- nlp.nap),
     ny o= p(nlp.nb o- nlp.nap).nlen2p(),
     nd o= nx o* nx o- ny o* p((nlp.na o- nap.nop).nlen2p() o- nap.nr o* nap.nrp);
  kif p(nsignp(ndp) o< l+m+mi0p) kreturn n+nbfalsep;
  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);
  np1 o= np o+ ndeltap; np2 o= np o- ndeltap;
  kreturn n+nbtruep;
p}
c+c1//圆与圆的交面积
nDB n+nfareaCCp(kconst nCircleo& nc1p, kconst nCircleo& nc2p) p{
  nDB nd o= p(nc1p.no o- nc2p.nop).nlenp();
  kif p(nsignp(nd o- p(nc1p.nr o+ nc2p.nrp)) o>= l+m+mi0p) kreturn l+m+mi0p;
  kif p(nsignp(nd o- nstdo::nabsp(nc1p.nr o- nc2p.nrp)) o<= l+m+mi0p) p{
    nDB nr o= nstdo::nminp(nc1p.nrp, nc2p.nrp);
    kreturn nr o* nr o* nPIp;
  p}
  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),
    nt1 o= nacosp(nx o/ nc1p.nrp), nt2 o= nacosp((nd o- nxp) o/ nc2p.nrp);
  kreturn nc1p.nr o* nc1p.nr o* nt1 o+ nc2p.nr o* nc2p.nr o* nt2 o- nd o* nc1p.nr o*
  ↪ nsinp(nt1p);
p}
c+c1// 圆与圆交点
k+ktbool n+nfisCCp(nCircle nap, nCircle nbp, nPo& np1p, nPo& np2p) p{
  nDB ns1 o= p(nap.no o- nbp.nop).nlenp();
  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;
  nDB ns2 o= p(nap.nr o* nap.nr o- nbp.nr o* nbp.nrp) o/ ns1p;
  nDB naa o= p(ns1 o+ ns2p) o* l+m+mf0.5p, nbb o= p(ns1 o- ns2p) o* l+m+mf0.5p;
  nP no o= p(nbp.no o- nap.nop) o* p(naa o/ p(naa o+ nbbp)) o+ nap.nop;
  nP ndelta o= p(nbp.no o- nap.nop).nunitp().nturn90p() o* nmsqrtp(nap.nr o* nap.nr o- naa o*
  ↪ naap);
  np1 o= no o+ ndeltap, np2 o= no o- ndeltap;
  kreturn n+nbtruep;
p}
c+c1// 求点到圆的切点，按关于点的顺时针方向返回两个点
k+ktbool n+nftanCPp(kconst nCircle o&ncp, kconst nPoint o&np0p, nPoint o&np1p, nPoint o&np2p)
  ↪ p{
  k+ktdouble nx o= p(np0 o- ncp.nop).nlen2p(), nd o= nx o- ncp.nr o* ncp.nrp;
  kif p(nd o< nepsp) kreturn n+nbfalsep; c+c1// 点在圆上认为没有切点
  nPoint np o= p(np0 o- ncp.nop) o* p(ncp.nr o* ncp.nr o/ nxp);
  nPoint ndelta o= p((np0 o- ncp.nop) o* p(o-ncp.nr o* nsqrtp(ndp) o/ nxp)).nturn90p();
  np1 o= ncp.no o+ np o+ ndeltap;
  np2 o= ncp.no o+ np o- ndeltap;
  kreturn n+nbtruep;
p}
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
nvectoro<nLineo> nextanCCp(kconst nCircle o&nc1p, kconst nCircle o&nc2p) p{
  nvectoro<nLineo> nretp;
  kif p(nsignp(nc1p.nr o- nc2p.nrp) o== l+m+mi0p) p{
    nPoint ndir o= nc2p.no o- nc1p.nop;
```

```
    ndir o= p(ndir o* p(nc1p.nr o/ ndirp.nlenp())).nturn90p();
    nretp.npush_backp(nLinep(nc1p.no o+ ndirp, nc2p.no o+ ndirp));
    nretp.npush_backp(nLinep(nc1p.no o- ndirp, nc2p.no o- ndirp));
  p} kelse p{
    nPoint np o= p(nc1p.no o* o-nc2p.nr o+ nc2p.no o* nc1p.nrp) o/ p(nc1p.nr o- nc2p.nrp);
    nPoint np1p, np2p, nq1p, nq2p;
    kif p(ntanCPp(nc1p, npp, np1p, np2p) o&& ntanCPp(nc2p, npp, nq1p, nq2p)) p{
      kif p(nc1p.nr o< nc2p.nrp) nswapp(np1p, np2p), nswapp(nq1p, nq2p);
      nretp.npush_backp(nLinep(np1p, nq1p));
      nretp.npush_backp(nLinep(np2p, nq2p));
    p}
  p}
  kreturn nretp;
p}
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
nstdo::nvectoro<nLineo> nintanCCp(kconst nCircle o&nc1p, kconst nCircle o&nc2p) p{
  nstdo::nvectoro<nLineo> nretp;
  nPoint np o= p(nc1p.no o* nc2p.nr o+ nc2p.no o* nc1p.nrp) o/ p(nc1p.nr o+ nc2p.nrp);
  nPoint np1p, np2p, nq1p, nq2p;
  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
    nretp.npush_backp(nLinep(np1p, nq1p));
    nretp.npush_backp(nLinep(np2p, nq2p));
  p}
  kreturn nretp;
p}
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
  k+ktint nret o= l+m+mi0p, nn o= npolygonp.nsizep();
  kforp(k+ktint ni o= l+m+mi0p; ni o< nnp; o++ nip) p{
    nPoint nu o= npolygonp[nip], nv o= npolygonp[(ni o+ l+m+mi1p) o% nnp];
    kif p(nonSegp(nLinep(nup, nvp), npp)) kreturn n+nbtruep;  c+c1// Here I guess.
    kif p(nsignp(nup.ny o- nvp.nyp) o<= l+m+mi0p) nswapp(nup, nvp);
    kif p(nsignp(npp.ny o- nup.nyp) o> l+m+mi0 o|| nsignp(npp.ny o- nvp.nyp) o<= l+m+mi0p) kcontinuep;
    nret o+= nsignp(ndetp(npp, nvp, nup)) o> l+m+mi0p;
  p}
  kreturn nret o& l+m+mi1p;
p}
c+c1// 用半平面 (q1,q2) 的逆时针方向去切凸多边形
nstdo::nvectoro<nPointo> nconvexCutp(kconst nstdo::nvectoro<nPointo>&npsp, nPoint nq1p, nPoint nq2p) p{
  nstdo::nvectoro<nPointo> nqsp; k+ktint nn o= npsp.nsizep();
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; o++nip) p{
    nPoint np1 o= npsp[nip], np2 o= npsp[(ni o+ l+m+mi1p) o% nnp];
    k+ktint nd1 o= ncrossOpp(nq1p,nq2p,np1p), nd2 o= ncrossOpp(nq1p,nq2p,np2p);
    kif p(nd1 o>= l+m+mi0p) nqsp.npush_backp(np1p);
    kif p(nd1 o* nd2 o< l+m+mi0p) nqsp.npush_backp(nisSSp(np1p, np2p, nq1p, nq2p));
  p}
  kreturn nqsp;
p}
c+c1// 求凸包
nstdo::nvectoro<nPointo> nconvexHullp(nstdo::nvectoro<nPointo> npsp) p{
  k+ktint nn o= npsp.nsizep(); kif p(nn o<= l+m+mi1p) kreturn npsp;
  nstdo::nsortp(npsp.nbeginp(), npsp.nendp());
  nstdo::nvectoro<nPointo> nqsp;
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nqsp.npush_backp(npsp[ni o++p]))
    kwhile p(nqsp.nsizep() o> l+m+mi1 o&& nsignp(ndetp(nqsp[nqsp.nsizep() o- l+m+mi2p], nqsp.nbackp(), npsp[nip])) o<= l+m+mi0p)
      nqsp.npop_backp();
  kfor p(k+ktint ni o= nn o- l+m+mi2p, nt o= nqsp.nsizep(); ni o>= l+m+mi0p; nqsp.npush_backp(npsp[ni o--p]))
    kwhile p((k+ktintp)nqsp.nsizep() o> nt o&& nsignp(ndetp(nqsp[nqsp.nsizep() o- l+m+mi2p], nqsp.nbackp(), npsp[nip])) o<= l+m+mi0p)
      nqsp.npop_backp();
  kreturn nqsp;
```

凸包

```
c+c1// 凸包中的点按逆时针方向
kstruct nConvex p{
  k+ktint nnp;
  nstdo::nvectoro<nPointo> nap, nupperp, nlowerp;
  k+ktvoid n+nfmake_shellp(kconst nstdo::nvectoro<nPointo>& npp,
      nstdo::nvectoro<nPointo>& nshellp) p{  c+c1// p needs to be sorted.
    nclearp(nshellp); k+ktint nn o= npp.nsizep();
    kfor p(k+ktint ni o= l+m+mi0p, nj o= l+m+mi0p; ni o< nnp; nio++p, njo++p) p{
      kfor p(; nj o>= l+m+mi2 o&& nsignp(ndetp(nshellp[njo-l+m+mi1p] o- nshellp[njo-l+m+mi2p],
              npp[nip] o- nshellp[njo-l+m+mi2p])) o<= l+m+mi0p; o--njp) nshellp.npop_backp();
      nshellp.npush_backp(npp[nip]);
    p}
  p}
  k+ktvoid n+nfmake_convexp() p{
    nstdo::nsortp(nap.nbeginp(), nap.nendp());
    nmake_shellp(nap, nlowerp);
    nstdo::nreversep(nap.nbeginp(), nap.nendp());
    nmake_shellp(nap, nupperp);
    na o= nlowerp; nap.npop_backp();
    nap.ninsertp(nap.nendp(), nupperp.nbeginp(), nupperp.nendp());
    kif p((k+ktintp)nap.nsizep() o>= l+m+mi2p) nap.npop_backp();
    nn o= nap.nsizep();
  p}
  k+ktvoid n+nfinitp(kconst nstdo::nvectoro<nPointo>& n_ap) p{
    nclearp(nap); na o= n_ap; nn o= nap.nsizep();
    nmake_convexp();
  p}
  k+ktvoid n+nfreadp(k+ktint n_np) p{  c+c1// Won't make convex.
    nclearp(nap); nn o= n_np; nap.nresizep(nnp);
    kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p)
      nap[nip].nreadp();
  p}
  nstdo::npairo<nDBp, k+ktinto> nget_tangentp(
      kconst nstdo::nvectoro<nPointo>& nconvexp, kconst nPointo& nvecp) p{
    k+ktint nl o= l+m+mi0p, nr o= p(k+ktintp)nconvexp.nsizep() o- l+m+mi2p;
    nassertp(nr o>= l+m+mi0p);
    kfor p(; nl o+ l+m+mi1 o< nrp; p) p{
      k+ktint nmid o= p(nl o+ nrp) o/ l+m+mi2p;
      kif p(nsignp(ndetp(nconvexp[nmid o+ l+m+mi1p] o- nconvexp[nmidp], nvecp)) o> l+m+mi0p)
```

```
      nr o= nmidp;
    kelse nl o= nmidp;
  p}
  kreturn nstdo::nmaxp(nstdo::nmake_pairp(ndetp(nvecp, nconvexp[nrp]), nrp),
      nstdo::nmake_pairp(ndetp(nvecp, nconvexp[l+m+mi0p]), l+m+mi0p));
 p}
 k+ktint nbinary_searchp(nPoint nup, nPoint nvp, k+ktint nlp, k+ktint nrp) p{
  k+ktint ns1 o= nsignp(ndetp(nv o- nup, nap[nl o% nnp] o- nup));
  kfor p(; nl o+ l+m+mi1 o< nrp; p) p{
    k+ktint nmid o= p(nl o+ nrp) o/ l+m+mi2p;
    k+ktint nsmid o= nsignp(ndetp(nv o- nup, nap[nmid o% nnp] o- nup));
    kif p(nsmid o== ns1p) nl o= nmidp;
    kelse nr o= nmidp;
  p}
  kreturn nl o% nnp;
 p}
 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
 k+ktint nget_tangentp(nPoint nvecp) p{
  nstdo::npairo<nDBp, k+ktinto> nret o= nget_tangentp(nupperp, nvecp);
  nretp.nsecond o= p(nretp.nsecond o+ p(k+ktintp)nlowerp.nsizep() o- l+m+mi1p) o% nnp;
  nret o= nstdo::nmaxp(nretp, nget_tangentp(nlowerp, nvecp));
  kreturn nretp.nsecondp;
 p}
 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
 k+ktbool nget_intersectionp(nPoint nup, nPoint nvp, k+ktint o&ni0p, k+ktint o&ni1p) p{
  k+ktint np0 o= nget_tangentp(nu o- nvp), np1 o= nget_tangentp(nv o- nup);
  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{
    kif p(np0 o> np1p) nstdo::nswapp(np0p, np1p);
    ni0 o= nbinary_searchp(nup, nvp, np0p, np1p);
    ni1 o= nbinary_searchp(nup, nvp, np1p, np0 o+ nnp);
    kreturn n+nbtruep;
  p}
  kelse kreturn n+nbfalsep;
 p}
p};
```

三角形的心

```
nPoint n+nfinCenterp(kconst nPoint o&nAp, kconst nPoint o&nBp, kconst nPoint o&nCp) p{ c+c1//
  ↪ 内心
 k+ktdouble na o= p(nB o- nCp).nlenp(), nb o= p(nC o- nAp).nlenp(), nc o= p(nA o-
  ↪ nBp).nlenp(),
  ns o= nfabsp(ndetp(nB o- nAp, nC o- nAp)),
  nr o= ns o/ npp;
 kreturn p(nA o* na o+ nB o* nb o+ nC o* ncp) o/ p(na o+ nb o+ ncp);
p}
nPoint n+nfcircumCenterp(kconst nPoint o&nap, kconst nPoint o&nbp, kconst nPoint o&ncp) p{
  ↪ c+c1// 外心
 nPoint nbb o= nb o- nap, ncc o= nc o- nap;
 k+ktdouble ndb o= nbbp.nlen2p(), ndc o= nccp.nlen2p(), nd o= l+m+mi2 o* ndetp(nbbp, nccp);
 kreturn na o- nPointp(nbbp.ny o* ndc o- nccp.ny o* ndbp, nccp.nx o* ndb o- nbbp.nx o* ndcp)
  ↪ o/ ndp;
p}
nPoint n+nfothroCenterp(kconst nPoint o&nap, kconst nPoint o&nbp, kconst nPoint o&ncp) p{
  ↪ c+c1// 垂心
 nPoint nba o= nb o- nap, nca o= nc o- nap, nbc o= nb o- ncp;
 k+ktdouble nY o= nbap.ny o* ncap.ny o* nbcp.nyp,
      nA o= ncap.nx o* nbap.ny o- nbap.nx o* ncap.nyp,
      nx0 o= p(nY o+ ncap.nx o* nbap.ny o* nbp.nx o- nbap.nx o* ncap.ny o* ncp.nxp) o/ nAp,
      ny0 o= o-nbap.nx o* p(nx0 o- ncp.nxp) o/ nbap.ny o+ ncap.nyp;
 kreturn nPointp(nx0p, ny0p);
p}
```

半平面交

```
kstruct nPoint p{
 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);}
p};
kstruct nLine p{
 k+ktbool nincludep(kconst nPoint o&npp) kconst p{ kreturn nsignp(ndetp(nb o- nap, np o-
  ↪ nap)) o> l+m+mi0p; p}
 nLine npushp() kconstp{ c+c1// 将半平面向外推 eps
   kconst k+ktdouble neps o= l+m+mf1e-6p;
   nPoint ndelta o= p(nb o- nap).nturn90p().nnormp() o* nepsp;
   kreturn n+nfLinep(na o- ndeltap, nb o- ndeltap);
 p}
p};
k+ktbool n+nfsameDirp(kconst nLine o&nl0p, kconst nLine o&nl1p) p{ kreturn nparallelp(nl0p,
  ↪ nl1p) o&& nsignp(ndotp(nl0p.nb o- nl0p.nap, nl1p.nb o- nl1p.nap)) o== l+m+mi1p; p}
k+ktbool koperator o< p(kconst nPoint o&nap, kconst nPoint o&nbp) p{
 kif p(nap.nquadp() o!= nbp.nquadp()) p{
   kreturn nap.nquadp() o< nbp.nquadp();
 p} kelse p{
   kreturn nsignp(ndetp(nap, nbp)) o> l+m+mi0p;
 p}
p}
k+ktbool koperator o< p(kconst nLine o&nl0p, kconst nLine o&nl1p) p{
 kif p(nsameDirp(nl0p, nl1p)) p{
   kreturn nl1p.nincludep(nl0p.nap);
 p} kelse p{
   kreturn p(nl0p.nb o- nl0p.nap) o< p(nl1p.nb o- nl1p.nap);
 p}
p}
k+ktbool ncheckp(kconst nLine o&nup, kconst nLine o&nvp, kconst nLine o&nwp) p{ kreturn
  ↪ nwp.nincludep(nintersectp(nup, nvp)); p}
nvectoro<nPointo> nintersectionp(nvectoro<nLineo> o&nlp) p{
 nsortp(nlp.nbeginp(), nlp.nendp());
 ndequeo<nLineo> nqp;
 kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nlp.nsizep(); o++nip) p{
   kif p(ni o&& nsameDirp(nlp[nip], nlp[ni o- l+m+mi1p])) p{
     kcontinuep;
   p}
   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();
   kwhile p(nqp.nsizep() o> l+m+mi1 o&& o!ncheckp(nqp[l+m+mi1p], nqp[l+m+mi0p], nlp[nip]))
  ↪ nqp.npop_frontp();
   nqp.npush_backp(nlp[nip]);
```

```
  p}
  kwhile p(nqp.nsizep() o> l+m+mi2 o&& o!ncheckp(nqp[nqp.nsizep() o- l+m+mi2p],
   ↪ nqp[nqp.nsizep() o- l+m+mi1p], nqp[l+m+mi0p])) nqp.npop_backp();
  kwhile p(nqp.nsizep() o> l+m+mi2 o&& o!ncheckp(nqp[l+m+mi1p], nqp[l+m+mi0p],
   ↪ nqp[nqp.nsizep() o- l+m+mi1p])) nqp.npop_frontp();
  nvectoro<nPointo> nretp;
  kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nqp.nsizep(); o++nip)
   ↪ nretp.npush_backp(nintersectp(nqp[nip], nqp[(ni o+ l+m+mi1p) o% nqp.nsizep()]));
  kreturn nretp;
p}
```

圆交面积及重心

```
kstruct nEvent p{
  nPoint npp;
  k+ktdouble nangp;
  k+ktint ndeltap;
  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{}
p};
k+ktbool koperator o< p(kconst nEvent o&nap, kconst nEvent o&nbp) p{
  kreturn nap.nang o< nbp.nangp;
p}
k+ktvoid naddEventp(kconst nCircle o&nap, kconst nCircle o&nbp, nvectoro<nEvento> o&nevtp,
   ↪ k+ktint o&ncntp) p{
  k+ktdouble nd2 o= p(nap.no o- nbp.nop).nlen2p(),
       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,
       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));
  nPoint nd o= nbp.no o- nap.nop, np o= ndp.nrotatep(nPI o/ l+m+mi2p),
      nq0 o= nap.no o+ nd o* ndRatio o+ np o* npRatiop,
      nq1 o= nap.no o+ nd o* ndRatio o- np o* npRatiop;
  k+ktdouble nang0 o= p(nq0 o- nap.nop).nangp(),
       nang1 o= p(nq1 o- nap.nop).nangp();
  nevtp.npush_backp(nEventp(nq1p, nang1p, l+m+mi1p));
  nevtp.npush_backp(nEventp(nq0p, nang0p, o-l+m+mi1p));
  ncnt o+= nang1 o> nang0p;
p}
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}
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}
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}
nCircle ncp[nNp];
k+ktdouble nareap[nNp];  c+c1// area[k] -> area of intersections >= k.
nPoint ncentroidp[nNp];
k+ktbool nkeepp[nNp];
k+ktvoid n+nfaddp(k+ktint ncntp, nDB nap, nPoint ncp) p{
  nareap[ncntp] o+= nap;
  ncentroidp[ncntp] o= ncentroidp[ncntp] o+ nc o* nap;
p}
k+ktvoid n+nfsolvep(k+ktint nCp) p{
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nCp; o++ nip) p{
       nareap[nip] o= l+m+mi0p;
       ncentroidp[nip] o= nPointp(l+m+mi0p, l+m+mi0p);
  p}
 kfor p(k+ktint ni o= l+m+mi0p; ni o< nCp; o++nip) p{
  k+ktint ncnt o= l+m+mi1p;
  nvectoro<nEvento> nevtp;
  kfor p(k+ktint nj o= l+m+mi0p; nj o< nip; o++njp) kif p(nissamep(ncp[nip], ncp[njp]))
   ↪ o++ncntp;
  kfor p(k+ktint nj o= l+m+mi0p; nj o< nCp; o++njp) p{
    kif p(nj o!= ni o&& o!nissamep(ncp[nip], ncp[njp]) o&& noverlapp(ncp[njp], ncp[nip]))
   ↪ p{
      o++ncntp;
    p}
  p}
  kfor p(k+ktint nj o= l+m+mi0p; nj o< nCp; o++njp) p{
    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{
      naddEventp(ncp[nip], ncp[njp], nevtp, ncntp);
    p}
  p}
  kif p(nevtp.nsizep() o== l+m+mi0up) p{
    naddp(ncntp, nPI o* ncp[nip].nr o* ncp[nip].nrp, ncp[nip].nop);
  p} kelse p{
    nsortp(nevtp.nbeginp(), nevtp.nendp());
    nevtp.npush_backp(nevtp.nfrontp());
    kfor p(k+ktint nj o= l+m+mi0p; nj o+ l+m+mi1 o< p(k+ktintp)nevtp.nsizep(); o++njp) p{
      ncnt o+= nevtp[njp].ndeltap;
      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);
      k+ktdouble nang o= nevtp[nj o+ l+m+mi1p].nang o- nevtp[njp].nangp;
      kif p(nang o< l+m+mi0p) p{
        nang o+= nPI o* l+m+mi2p;
      p}
              kif p(nsignp(nangp) o== l+m+mi0p) kcontinuep;
              naddp(ncntp, nang o* ncp[nip].nr o* ncp[nip].nr 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));
      naddp(ncntp, o-nsinp(nangp) o* ncp[nip].nr o* ncp[nip].nr o/ l+m+mi2p, p(ncp[nip].no
   ↪ o+ nevtp[njp].np o+ nevtp[nj o+ l+m+mi1p].npp) o/ l+m+mi3p);
    p}
  p}
 p}
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nCp; o++ nip)
  kif p(nsignp(nareap[nip])) p{
    ncentroidp[nip] o= ncentroidp[nip] o/ nareap[nip];
  p}
p}
```

三维向量绕轴旋转

```
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
nPoint n+nfrotatep(kconst nPointo& nsp, kconst nPointo& naxisp, nDB nwp) p{
  nDB nx o= naxisp.nxp, ny o= naxisp.nyp, nz o= naxisp.nzp;
  nDB ns1 o= nx o* nx o+ ny o* ny o+ nz o* nzp, nss1 o= nmsqrtp(ns1p),
     ncosw o= ncosp(nwp), nsinw o= nsinp(nwp);
  nDB nap[l+m+mi4p][l+m+mi4p];
```

```
  nmemsetp(nap, l+m+mi0p, ksizeof nap);
  nap[l+m+mi3p][l+m+mi3p] o= l+m+mi1p;
  nap[l+m+mi0p][l+m+mi0p] o= p((ny o* ny o+ nz o* nzp) o* ncosw o+ nx o* nxp) o/ ns1p;
  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;
  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;
  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;
  nap[l+m+mi1p][l+m+mi1p] o= p((nx o* nx o+ nz o* nzp) o* ncosw o+ ny o* nyp) o/ ns1p;
  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;
  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;
  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;
  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;
  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};
  kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi4p; o++ nip)
    kfor p(k+ktint nj o= l+m+mi0p; nj o< l+m+mi4p; o++ njp)
      nansp[nip] o+= nap[njp][nip] o* ncp[njp];
  kreturn nPointp(nansp[l+m+mi0p], nansp[l+m+mi1p], nansp[l+m+mi2p]);
p}
```

三维凸包

```
k+kr__inline nP n+nfcrossp(kconst nPo& nap, kconst nPo& nbp) p{
  kreturn nPp(
      nap.ny o* nbp.nz o- nap.nz o* nbp.nyp,
      nap.nz o* nbp.nx o- nap.nx o* nbp.nzp,
      nap.nx o* nbp.ny o- nap.ny o* nbp.nx
        p);
p}

k+kr__inline nDB n+nfmixp(kconst nPo& nap, kconst nPo& nbp, kconst nPo& ncp) p{
  kreturn ndotp(ncrossp(nap, nbp), ncp);
p}

k+kr__inline nDB n+nfvolumep(kconst nPo& nap, kconst nPo& nbp, kconst nPo& ncp, kconst nPo&
   ↪ ndp) p{
  kreturn nmixp(nb o- nap, nc o- nap, nd o- nap);
p}

kstruct nFace p{
  k+ktint nap, nbp, ncp;
  k+kr__inline n+nfFacep() p{}
  k+kr__inline n+nfFacep(k+ktint n_ap, k+ktint n_bp, k+ktint n_cp)o:
    nap(n_ap), nbp(n_bp), ncp(n_cp) p{}
  k+kr__inline nDB n+nfareap() kconst p{
    kreturn l+m+mf0.5 o* ncrossp(npp[nbp] o- npp[nap], npp[ncp] o- npp[nap]).nlenp();
  p}
  k+kr__inline nP n+nfnormalp() kconst p{
    kreturn ncrossp(npp[nbp] o- npp[nap], npp[ncp] o- npp[nap]).nunitp();
  p}
  k+kr__inline nDB n+nfdisp(kconst nPo& np0p) kconst p{
    kreturn ndotp(nnormalp(), np0 o- npp[nap]);
  p}
p};

nstdo::nvectoro<nFaceo> nfacep, ntmpp;  c+c1// Should be O(n).
k+ktint nmarkp[nNp][nNp], nTimep, nnp;

k+kr__inline k+ktvoid n+nfaddp(k+ktint nvp) p{
  o++ nTimep;
  nclearp(ntmpp);
  kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nfacep.nsizep(); o++ nip) p{
    k+ktint na o= nfacep[nip].nap, nb o= nfacep[nip].nbp, nc o= nfacep[nip].ncp;
    kif p(nsignp(nvolumep(npp[nvp], npp[nap], npp[nbp], npp[ncp])) o> l+m+mi0p) p{
      nmarkp[nap][nbp] o= nmarkp[nbp][nap] o= nmarkp[nap][ncp] o=
        nmarkp[ncp][nap] o= nmarkp[nbp][ncp] o= nmarkp[ncp][nbp] o= nTimep;
    p}
    kelse p{
      ntmpp.npush_backp(nfacep[nip]);
    p}
  p}
  nclearp(nfacep); nface o= ntmpp;
  kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)ntmpp.nsizep(); o++ nip) p{
    k+ktint na o= nfacep[nip].nap, nb o= nfacep[nip].nbp, nc o= nfacep[nip].ncp;
    kif p(nmarkp[nap][nbp] o== nTimep) nfacep.nemplace_backp(nvp, nbp, nap);
    kif p(nmarkp[nbp][ncp] o== nTimep) nfacep.nemplace_backp(nvp, ncp, nbp);
    kif p(nmarkp[ncp][nap] o== nTimep) nfacep.nemplace_backp(nvp, nap, ncp);
    nassertp(nfacep.nsizep() o< l+m+mi500up);
  p}
p}

k+ktvoid n+nfreorderp() p{
  kfor p(k+ktint ni o= l+m+mi2p; ni o< nnp; o++ nip) p{
    nP ntmp o= ncrossp(npp[nip] o- npp[l+m+mi0p], npp[nip] o- npp[l+m+mi1p]);
    kif p(nsignp(ntmpp.nlenp())) p{
      nstdo::nswapp(npp[nip], npp[l+m+mi2p]);
      kfor p(k+ktint nj o= l+m+mi3p; nj o< nnp; o++ njp)
        kif p(nsignp(nvolumep(npp[l+m+mi0p], npp[l+m+mi1p], npp[l+m+mi2p], npp[njp]))) p{
          nstdo::nswapp(npp[njp], npp[l+m+mi3p]);
          kreturnp;
        p}
    p}
  p}
p}

k+ktvoid n+nfbuild_convexp() p{
  nreorderp();
  nclearp(nfacep);
  nfacep.nemplace_backp(l+m+mi0p, l+m+mi1p, l+m+mi2p);
  nfacep.nemplace_backp(l+m+mi0p, l+m+mi2p, l+m+mi1p);
  kfor p(k+ktint ni o= l+m+mi3p; ni o< nnp; o++ nip)
    naddp(nip);
p}
```

数论

$O(m^2 \log n)$ 求线性递推数列第 n 项

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

```
k+ktvoid n+nflinear_recurrencep(k+ktlong k+ktlong nnp, k+ktint nmp, k+ktint nap[], k+ktint
  ↪ ncp[], k+ktint npp) p{
  k+ktlong k+ktlong nvp[nMp] o= p{l+m+mi1 o% npp}, nup[nM o<< l+m+mi1p], nmsk o= o!!nnp;
  kforp(k+ktlong k+ktlong nip(nnp); ni o> l+m+mi1p; ni o>>= l+m+mi1p) p{
    nmsk o<<= l+m+mi1p;
  p}
  kforp(k+ktlong k+ktlong nxp(l+m+mi0p); nmskp; nmsk o>>= l+m+mi1p, nx o<<= l+m+mi1p) p{
    nfill_np(nup, nm o<< l+m+mi1p, l+m+mi0p);
    k+ktint nbp(o!!p(nn o& nmskp));
    nx o|= nbp;
    kifp(nx o< nmp) p{
      nup[nxp] o= l+m+mi1 o% npp;
    p}kelse p{
      kforp(k+ktint nip(l+m+mi0p); ni o< nmp; nio++p) p{
        kforp(k+ktint njp(l+m+mi0p), ntp(ni o+ nbp); nj o< nmp; njo++p, nto++p) p{
          nup[ntp] o= p(nup[ntp] o+ nvp[nip] o* nvp[njp]) o% npp;
        p}
      p}
      kforp(k+ktint nip((nm o<< l+m+mi1p) o- l+m+mi1p); ni o>= nmp; nio--p) p{
        kforp(k+ktint njp(l+m+mi0p), ntp(ni o- nmp); nj o< nmp; njo++p, nto++p) p{
          nup[ntp] o= p(nup[ntp] o+ ncp[njp] o* nup[nip]) o% npp;
        p}
      p}
    p}
    ncopyp(nup, nu o+ nmp, nvp);
  p}
  c+c1//a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
  kforp(k+ktint nip(nmp); ni o< l+m+mi2 o* nmp; nio++p) p{
    nap[nip] o= l+m+mi0p;
    kforp(k+ktint njp(l+m+mi0p); nj o< nmp; njo++p) p{
      nap[nip] o= p(nap[nip] o+ p(k+ktlong k+ktlongp)ncp[njp] o* nap[ni o+ nj o- nmp]) o%
   ↪ npp;
    p}
  p}
  kforp(k+ktint njp(l+m+mi0p); nj o< nmp; njo++p) p{
    nbp[njp] o= l+m+mi0p;
    kforp(k+ktint nip(l+m+mi0p); ni o< nmp; nio++p) p{
      nbp[njp] o= p(nbp[njp] o+ nvp[nip] o* nap[ni o+ njp]) o% npp;
    p}
  p}
  kforp(k+ktint njp(l+m+mi0p); nj o< nmp; njo++p) p{
    nap[njp] o= nbp[njp];
  p}
p}
```

求逆元

```
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{
  kif p(nb o== l+m+mi0p) p{
    nx o= l+m+mi1p;
    ny o= l+m+mi0p;
    kreturnp;
  p}
  k+ktlong k+ktlong nxxp, nyyp;
  nex_gcdp(nbp, na o% nbp, nxxp, nyyp);
  ny o= nxx o- na o/ nb o* nyyp;
  nx o= nyyp;
p}

k+ktlong k+ktlong n+nfinvp(k+ktlong k+ktlong nxp, k+ktlong k+ktlong nMODNp) p{
  k+ktlong k+ktlong ninv_xp, nyp;
  nex_gcdp(nxp, nMODNp, ninv_xp, nyp);
  kreturn p(ninv_x o% nMODN o+ nMODNp) o% nMODNp;
p}
```

中国剩余定理

```
c+c1// 返回 (ans, M), 其中 ans 是模 M 意义下的解
nstdo::npairo<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{
  k+ktlong k+ktlong nM o= l+m+mi1p, nans o= l+m+mi0p;
  k+ktint nn o= nmp.nsizep();
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) nM o*= nmp[nip];
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
    nans o= p(nans o+ p(nM o/ nmp[nip]) o* nap[nip] o% nM o* ninvp(nM o/ nmp[nip], nmp[nip]))
  ↪ o% nMp;  c+c1// 可能需要大整数相乘取模
  p}
  kreturn nstdo::nmake_pairp(nansp, nMp);
p}
c+c1// 模数不互质的情况
k+ktbool nsolvep(k+ktint nnp, nstdo::npairo<k+ktlong k+ktlongp, k+ktlong k+ktlongo>
  ↪ ninputp[],
                 nstdo::npairo<k+ktlong k+ktlongp, k+ktlong k+ktlongo> o&noutputp) p{
  noutput o= nstdo::nmake_pairp(l+m+mi1p, l+m+mi1p);
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; o++nip) p{
    k+ktlong k+ktlong nnumberp, nuselessp;
    c+c1// euclid(a, b, x, y)
    neuclidp(noutputp.nsecondp, ninputp[nip].nsecondp, nnumberp, nuselessp);
    k+ktlong k+ktlong ndivisor o= nstdo::n__gcdp(noutputp.nsecondp, ninputp[nip].nsecondp);
    kif p((ninputp[nip].nfirst o- noutputp.nfirstp) o% ndivisorp) kreturn n+nbfalsep;
    nnumber o*= p(ninputp[nip].nfirst o- noutputp.nfirstp) o/ ndivisorp;
    nfixp(nnumberp, ninputp[nip].nsecondp);  c+c1// fix 成正的
    noutputp.nfirst o+= noutputp.nsecond o* nnumberp;
    noutputp.nsecond o*= ninputp[nip].nsecond o/ ndivisorp;
    nfixp(noutputp.nfirstp, noutputp.nsecondp);
  p}
  kreturn n+nbtruep;
p}
```

魔法 CRT

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

```
      kif p(nt o< l+m+mi0p) nt o+= nmodp[nip];
      nxp[nip] o= l+m+mi1LL o* nt o* nInvp[njp][nip] o% nmodp[nip];
    p}
  p}
  k+ktint nsum o= l+m+mi1p, nret o= nxp[l+m+mi0p] o% nMODp;
  kfor p(k+ktint ni o= l+m+mi1p; ni o< nNp; ni o++p) p{
    nsum o= l+m+mi1LL o* nsum o* nmodp[ni o- l+m+mi1p] o% nMODp;
    nret o+= l+m+mi1LL o* nxp[nip] o* nsum o% nMODp;
    kif p(nret o>= nMODp) nret o-= nMODp;
  p}
  kreturn nretp;
p}
kfor p(k+ktint ni o= l+m+mi0p; ni o< nNp; ni o++p)
  kfor p(k+ktint nj o= ni o+ l+m+mi1p; nj o< nNp; nj o++p) p{
    nInvp[nip][njp] o= nfpwp(nmodp[nip], nmodp[njp] o- l+m+mi2p, nmodp[njp]);
  p}
```

素性测试

```
k+ktint n+nfstrong_pseudo_primetestp(k+ktlong k+ktlong nnp,k+ktint nbasep) p{
    k+ktlong k+ktlong nn2o=nno-l+m+mi1p,nresp;
    k+ktint nso=l+m+mi0p;
    kwhilep(nn2o%l+m+mi2o==l+m+mi0p) nn2o>>=l+m+mi1p,nso++p;
    nreso=npowmodp(nbasep,nn2p,nnp);
    kifp((nreso==l+m+mi1p)o||p(nreso==nno-l+m+mi1p)) kreturn l+m+mi1p;
    nso--p;
    kwhilep(nso>=l+m+mi0p) p{
        nreso=nmulmodp(nresp,nresp,nnp);
        kifp(nreso==nno-l+m+mi1p) kreturn l+m+mi1p;
        nso--p;
    p}
    kreturn l+m+mi0p; c+c1// n is not a strong pseudo prime
p}
k+ktint n+nfisprimep(k+ktlong k+ktlong nnp) p{
  kstatic nLL
   ↪ 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};
  kstatic nLL
   ↪ nlimp[]o=p{l+m+mi4p,l+m+mi0p,l+m+mi1373653LLp,l+m+mi25326001LLp,l+m+mi25000000000LLp,l+m+mi2152302898747LLp,
   ↪ l+m+mi3474749660383LLp,l+m+mi341550071728321LLp,l+m+mi0p,l+m+mi0p,l+m+mi0p,l+m+mi0p};
  kifp(nno<l+m+mi2o||nno==l+m+mi3215031751LLp) kreturn l+m+mi0p;
  kforp(k+ktint nio=l+m+mi0p;nio<l+m+mi12p;o++nip){
    kifp(nno<nlimp[nip]) kreturn l+m+mi1p;
    kifp(nstrong_pseudo_primetestp(nnp,ntestNump[nip])o==l+m+mi0p) kreturn l+m+mi0p;
  p}
  kreturn l+m+mi1p;
p}
```

质因数分解

```
k+ktint nansnp; nLL nansp[l+m+mi1000p];
nLL n+nffuncp(nLL nxp,nLL nnp){ kreturnp(nmod_mulp(nxp,nxp,nnp)o+l+m+mi1p)o%nnp; p}
nLL n+nfPollardp(nLL nnp){
  nLL nip,nxp,nyp,npp;
  kifp(nRabin_Millerp(nnp)) kreturn nnp;
  kifp(o!p(nno&l+m+mi1p)) kreturn l+m+mi2p;
  kforp(nio=l+m+mi1p;nio<l+m+mi20p;nio++p){
    nxo=nip; nyo=nfuncp(nxp,nnp); npo=ngcdp(nyo-nxp,nnp);
    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;}
    kifp(npo==l+m+mi0o||npo==nnp) kcontinuep;
    kreturn npp;
  p}
p}
k+ktvoid n+nffactorp(nLL nnp){
  nLL nxp;
  nxo=nPollardp(nnp);
  kifp(nxo==nnp){ nansp[nansno++p]o=nxp; kreturnp; p}
  nfactorp(nxp), nfactorp(nno/nxp);
p}
```

线下整点

```
c+c1// \sum_{i=0}^{n-1} \lfloor \frac{a+bi}{m} \rfloor, n,m,a,b > 0
nLL n+nfsolvep(nLL nnp,nLL nap,nLL nbp,nLL nmp){
  kifp(nbo==l+m+mi0p) kreturn nno*p(nao/nmp);
  kifp(nao>=nmp) kreturn nno*p(nao/nmp)o+nsolvep(nnp,nao%nmp,nbp,nmp);
  kifp(nbo>=nmp) kreturn
   ↪ p(nno-l+m+mi1p)o*nno/l+m+mi2o*p(nbo/nmp)o+nsolvep(nnp,nap,nbo%nmp,nmp);
  kreturn nsolvep((nao+nbo*nnp)o/nmp,(nao+nbo*nnp)o%nmp,nmp,nbp);
p}
```

原根相关

1. 模 m 有原根的充要条件：$m = 2, 4, p^a, 2p^a$，其中 p 是奇素数；

2. 求任意数 p 原根的方法：对 $\phi(p)$ 因式分解，即 $\phi(p) = p_1^{r1} p_2^{r2} \cdots p_k^{rk}$，若恒成立：

$$g^{\frac{p-1}{g}} \neq 1 \pmod{p}$$

那么 g 就是 p 的原根。

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

快速傅里叶变换

```
k+ktint n+nfpreparep(k+ktint nnp) p{
  k+ktint nlen o= l+m+mi1p;
  kfor p(; nlen o<= l+m+mi2 o* nnp; nlen o<<= l+m+mi1p);
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nlenp; nio++p) p{
    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));
    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));
  p}
  kreturn nlenp;
p}
k+ktvoid n+nfDFTp(nComplex o*nap, k+ktint nnp, k+ktint nfp) p{
  kfor p(k+ktint ni o= l+m+mi0p, nj o= l+m+mi0p; ni o< nnp; nio++p) p{
    kif p(ni o> njp) nstdo::nswapp(nap[nip], nap[njp]);
    kfor p(k+ktint nt o= nn o>> l+m+mi1p; p(nj o^= ntp) o< ntp; nt o>>= l+m+mi1p);
  p}
```

```
  kfor p(k+ktint ni o= l+m+mi2p; ni o<= nnp; ni o<<= l+m+mi1p)
    kfor p(k+ktint nj o= l+m+mi0p; nj o< nnp; nj o+= nip)
      kfor p(k+ktint nk o= l+m+mi0p; nk o< p(ni o>> l+m+mi1p); nko++p) p{
        nComplex nA o= nap[nj o+ nkp];
        nComplex nB o= nep[nfp][nn o/ ni o* nkp] o* nap[nj o+ nk o+ p(ni o>> l+m+mi1p)];
        nap[nj o+ nkp] o= nA o+ nBp;
        nap[nj o+ nk o+ p(ni o>> l+m+mi1p)] o= nA o- nBp;
      p}
  kif p(nf o== l+m+mi1p) p{
    kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p)
      nap[nip].na o/= nnp;
  p}
p}
```

快速数论变换

```
c+c1// meminit(A, l, r) 是将数组 A 的 [l, r) 清 0。
c+c1// memcopy(target, source, l, r) 是将 source 的 [l, r) 复制到 target 的 [l, r)
c+cp#define meminit(A, l, r) memset(A + (l), 0, sizeof(*A) * ((r) - (l)))
c+cp#define memcopy(B, A, l, r) memcpy(B, A + (l), sizeof(*A) * ((r) - (l)))
k+ktvoid n+nfDFTp(k+ktint o*nap, k+ktint nnp, k+ktint nfp) p{ c+c1// 封闭形式，常数小
  ↪(10^7 跑 2.23 秒)
  kfor p(kregister k+ktint ni o= l+m+mi0p, nj o= l+m+mi0p; ni o< nnp; nio++p) p{
    kif p(ni o> njp) nstdo::nswapp(nap[nip], nap[njp]);
    kfor p(kregister k+ktint nt o= nn o>> l+m+mi1p; p(nj o^= ntp) o< ntp; nt o>>= l+m+mi1p);
  p}
  kfor p(kregister k+ktint ni o= l+m+mi2p; ni o<= nnp; ni o<<= l+m+mi1p) p{
    kstatic k+ktint nexpp[nMAXNp];
    nexpp[l+m+mi0p] o= l+m+mi1p; nexpp[l+m+mi1p] o= nfpmp(nPRTp, p(nMOD o- l+m+mi1p) o/ nip);
    kif p(nf o== l+m+mi1p) nexpp[l+m+mi1p] o= nfpmp(nexpp[l+m+mi1p], nMOD o- l+m+mi2p);
    kfor p(kregister k+ktint nk o= l+m+mi2p; nk o< p(ni o>> l+m+mi1p); nko++p) p{
      nexpp[nkp] o= l+m+mi1ll o* nexpp[nk o- l+m+mi1p] o* nexpp[l+m+mi1p] o% nMODp;
    p}
    kfor p(kregister k+ktint nj o= l+m+mi0p; nj o< nnp; nj o+= nip) p{
      kfor p(kregister k+ktint nk o= l+m+mi0p; nk o< p(ni o>> l+m+mi1p); nko++p) p{
        kregister k+ktint o&npA o= nap[nj o+ nkp], o&npB o= nap[nj o+ nk o+ p(ni o>>
  ↪l+m+mi1p)];
        kregister k+ktint nA o= npAp, nB o= l+m+mi1ll o* npB o* nexpp[nkp] o% nMODp;
        npA o= p(nA o+ nBp) o% nMODp;
        npB o= p(nA o- nB o+ nMODp) o% nMODp;
      p}
    p}
  p}
  kif p(nf o== l+m+mi1p) p{
    kregister k+ktint nrev o= nfpmp(nnp, nMOD o- l+m+mi2p, nMODp);
    kfor p(kregister k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
      nap[nip] o= l+m+mi1ll o* nap[nip] o* nrev o% nMODp;
    p}
  p}
p}
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
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
k+ktint n+nfCRTp(k+ktint o*nap) p{
  kstatic k+ktint nxp[l+m+mi3p];
  kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi3p; nio++p) p{
    nxp[nip] o= nap[nip];
    kfor p(k+ktint nj o= l+m+mi0p; nj o< nip; njo++p) p{
      k+ktint nt o= p(nxp[nip] o- nxp[njp] o+ nFFTp[nip] o-> nMODp) o% nFFTp[nip] o-> nMODp;
      kif p(nt o< l+m+mi0p) nt o+= nFFTp[nip] o-> nMODp;
      nxp[nip] o= l+m+mi1LL o* nt o* ninvp[njp][nip] o% nFFTp[nip] o-> nMODp;
    p}
  p}
  k+ktint nsum o= l+m+mi1p, nret o= nxp[l+m+mi0p] o% nMODp;
  kfor p(k+ktint ni o= l+m+mi1p; ni o< l+m+mi3p; ni o++p) p{
    nsum o= l+m+mi1LL o* nsum o* nFFTp[ni o- l+m+mi1p] o-> nMOD o% nMODp;
    nret o+= l+m+mi1LL o* nxp[nip] o* nsum o% nMODp;
    kifp(nret o>= nMODp) nret o-= nMODp;
  p}
  kreturn nretp;
p}
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
  kfor p(k+ktint nj o= l+m+mi0p; nj o< l+m+mi3p; njo++p)
    ninvp[nip][njp] o= ninversep(nFFTp[nip] o-> nMODp, nFFTp[njp] o-> nMODp);
```

自适应辛普森积分

```
knamespace nadaptive_simpson p{
  ktemplateo<ktypename nfunctiono>
  k+krinline k+ktdouble nareap(nfunction nfp, kconst k+ktdouble o&nleftp, kconst k+ktdouble
   ↪o&nrightp) p{
    k+ktdouble nmid o= p(nleft o+ nrightp) o/ l+m+mi2p;
    kreturn p(nright o- nleftp) o* p(nfp(nleftp) o+ l+m+mi4 o* nfp(nmidp) o+ nfp(nrightp)) o/
   ↪l+m+mi6p;
  p}

  ktemplateo<ktypename nfunctiono>
  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_sump) p{
    k+ktdouble nmid o= p(nleft o+ nrightp) o/ l+m+mi2p;
    k+ktdouble narea_left o= nareap(nfp, nleftp, nmidp);
    k+ktdouble narea_right o= nareap(nfp, nmidp, nrightp);
    k+ktdouble narea_total o= narea_left o+ narea_rightp;
    kif p(nfabsp(narea_total o- narea_sump) o<= l+m+mi15 o* nepsp) p{
      kreturn narea_total o+ p(narea_total o- narea_sump) o/ l+m+mi15p;
    p}
    kreturn nsimpsonp(nfp, nleftp, nrightp, neps o/ l+m+mi2p, narea_leftp) o+ nsimpsonp(nfp,
   ↪nmidp, nrightp, neps o/ l+m+mi2p, narea_rightp);
  p}

  ktemplateo<ktypename nfunctiono>
  k+krinline k+ktdouble nsimpsonp(nfunction nfp, kconst k+ktdouble o&nleftp, kconst
   ↪k+ktdouble o&nrightp, kconst k+ktdouble o&nepsp) p{
    kreturn nsimpsonp(nfp, nleftp, nrightp, nepsp, nareap(nfp, nleftp, nrightp));
  p}
p}
```

单纯形

```
kconst k+ktdouble neps o= l+m+mf1e-8p;
c+c1// max{c * x | Ax <= b, x >= 0} 的解，无解返回空的 vector，否则就是解.
nvectoro<k+ktdoubleo> nsimplexp(nvectoro<nvectoro<k+ktdoubleo> o> o&nAp,
    ↪ nvectoro<k+ktdoubleo> nbp, nvectoro<k+ktdoubleo> ncp) p{
  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;
  nvectoro<nvectoro<k+ktdoubleo> o> nDp(nn o+ l+m+mi2p, nvectoro<k+ktdoubleo>p(nm o+
    ↪ l+m+mi1p));
  nvectoro<k+ktinto> nixp(nn o+ nmp);
  kforp(k+ktint ni o= l+m+mi0p; ni o< nn o+ nmp; nio++p)  p{
    nixp[nip] o= nip;
  p}
  kforp(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
    kforp(k+ktint nj o= l+m+mi0p; nj o< nm o- l+m+mi1p; njo++p) p{
      nDp[nip][njp] o= o-nAp[nip][njp];
    p}
    nDp[nip][nm o- l+m+mi1p] o= l+m+mi1p;
    nDp[nip][nmp] o= nbp[nip];
    kif p(nDp[nrp][nmp] o> nDp[nip][nmp]) p{
      nr o= nip;
    p}
  p}

  kforp(k+ktint nj o= l+m+mi0p; nj o< nm o- l+m+mi1p; njo++p) p{
    nDp[nnp][njp] o= ncp[njp];
  p}
  nDp[nn o+ l+m+mi1p][nm o- l+m+mi1p] o= o-l+m+mi1p;
  kforp(k+ktdouble ndp; p;) p{
    kif p(nr o< nnp) p{
      nswapp(nixp[nsp], nixp[nr o+ nmp]);
      nDp[nrp][nsp] o= l+m+mf1. o/ nDp[nrp][nsp];
      kforp(k+ktint nj o= l+m+mi0p; nj o<= nmp; njo++p) p{
        kif p(nj o!= nsp) p{
          nDp[nrp][njp] o*= o-nDp[nrp][nsp];
        p}
      p}
      kforp(k+ktint ni o= l+m+mi0p; ni o<= nn o+ l+m+mi1p; nio++p) p{
        kif p(ni o!= nrp) p{
          kforp(k+ktint nj o= l+m+mi0p; nj o<= nmp; njo++p) p{
            kif p(nj o!= nsp) p{
              nDp[nip][njp] o+= nDp[nrp][njp] o* nDp[nip][nsp];
            p}
          p}
          nDp[nip][nsp] o*= nDp[nrp][nsp];
        p}
      p}
    p}
    nr o= o-l+m+mi1p, ns o= o-l+m+mi1p;
    kforp(k+ktint nj o= l+m+mi0p; nj o< nmp; njo++p) p{
      kif p(ns o< l+m+mi0 o|| nixp[nsp] o> nixp[njp]) p{
        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> nepsp) p{
          ns o= njp;
        p}
      p}
    p}
    kif p(ns o< l+m+mi0p) p{
      kbreakp;
    p}
    kforp(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p) p{
      kif p(nDp[nip][nsp] o< o-nepsp) p{
        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{

          nr o= nip;
        p}
      p}
    p}

    kif p(nr o< l+m+mi0p) p{
      kreturn nvectoro<k+ktdoubleo> p();
    p}
  p}
  kif p(nDp[nn o+ l+m+mi1p][nmp] o< o-nepsp) p{
    kreturn nvectoro<k+ktdoubleo> p();
  p}

  nvectoro<k+ktdoubleo> nxp(nm o- l+m+mi1p);
  kforp(k+ktint ni o= nmp; ni o< nn o+ nmp; nio++p) p{
    kif p(nixp[nip] o< nm o- l+m+mi1p) p{
      nxp[nixp[nip]] o= nDp[ni o- nmp][nmp];
    p}
  p}
  kreturn nxp;
p}
```

字符串

后缀数组

```
kconst k+ktint nMAXN o= nMAXL o* l+m+mi2 o+ l+m+mi1p;
k+ktint nap[nMAXNp], nxp[nMAXNp], nyp[nMAXNp], ncp[nMAXNp], nsap[nMAXNp], nrankp[nMAXNp],
    ↪ nheightp[nMAXNp];
k+ktvoid n+nfcalc_sap(k+ktint nnp) p{
  k+ktint nm o= nalphabetp, nk o= l+m+mi1p;
  nmemsetp(ncp, l+m+mi0p, ksizeofp(o*ncp) o* p(nm o+ l+m+mi1p));
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) ncp[nxp[nip] o= nap[nip]]o++p;
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nmp; o++nip) ncp[nip] o+= ncp[ni o- l+m+mi1p];
  kfor p(k+ktint ni o= nnp; nip; o--nip) nsap[ncp[nxp[nip]]o--p] o= nip;
  kfor p(; nk o<= nnp; nk o<<= l+m+mi1p) p{
    k+ktint ntot o= nkp;
    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;
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip)
      kif p(nsap[nip] o> nkp) nyp[o++ntotp] o= nsap[nip] o- nkp;
    nmemsetp(ncp, l+m+mi0p, ksizeofp(o*ncp) o* p(nm o+ l+m+mi1p));
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) ncp[nxp[nip]]o++p;
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nmp; o++nip) ncp[nip] o+= ncp[ni o- l+m+mi1p];
```

```
    kfor p(k+ktint ni o= nnp; nip; o--nip) nsap[ncp[nxp[nyp[nip]]]o--p] o= nyp[nip];
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) nyp[nip] o= nxp[nip];
    ntot o= l+m+mi1p; nxp[nsap[l+m+mi1p]] o= l+m+mi1p;
    kfor p(k+ktint ni o= l+m+mi2p; ni o<= nnp; o++nip) p{
      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;
      nxp[nsap[nip]] o= ntotp;
    p}
    kif p(ntot o== nnp) kbreakp; kelse nm o= ntotp;
  p}
p}
k+ktvoid n+nfcalc_heightp(k+ktint nnp) p{
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) nrankp[nsap[nip]] o= nip;
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip) p{
    nheightp[nrankp[nip]] o= nmaxp(l+m+mi0p, nheightp[nrankp[ni o- l+m+mi1p]] o- l+m+mi1p);
    kif p(nrankp[nip] o== l+m+mi1p) kcontinuep;
    k+ktint nj o= nsap[nrankp[nip] o- l+m+mi1p];
    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]];
  p}
p}
```

后缀自动机

```
kstatic kconst k+ktint nMAXL o= nMAXN o* l+m+mi2p;  c+c1// MAXN is original length
kstatic kconst k+ktint nalphabet o= l+m+mi26p;  c+c1// sometimes need changing
k+ktint nlp, nlastp, ncntp, ntransp[nMAXLp][nalphabetp], nparp[nMAXLp], nsump[nMAXLp],
   ↪ nseqp[nMAXLp], nmxlp[nMAXLp], nsizep[nMAXLp];  c+c1// mxl is maxlength, size is the size
   ↪ of right
k+ktchar nstrp[nMAXLp];
k+krinline k+ktvoid n+nfinitp() p{
  nl o= nstrlenp(nstr o+ l+m+mi1p); ncnt o= nlast o= l+m+mi1p;
  kfor p(k+ktint ni o= l+m+mi0p; ni o<= nl o* l+m+mi2p; o++nip) nmemsetp(ntransp[nip],
   ↪ l+m+mi0p, ksizeofp(ntransp[nip]));
  nmemsetp(nparp, l+m+mi0p, ksizeofp(o*nparp) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
  nmemsetp(nmxlp, l+m+mi0p, ksizeofp(o*nmxlp) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
  nmemsetp(nsizep, l+m+mi0p, ksizeofp(o*nsizep) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
p}
k+krinline k+ktvoid n+nfextendp(k+ktint nposp, k+ktint ncp) p{
  k+ktint np o= nlastp, nnp o= nlast o= o++ncntp;
  nmxlp[nnpp] o= nmxlp[npp] o+ l+m+mi1p; nsizep[nnpp] o= l+m+mi1p;
  kfor p(; np o&& o!ntransp[npp][ncp]; np o= nparp[npp]) ntransp[npp][ncp] o= nnpp;
  kif p(o!npp) nparp[nnpp] o= l+m+mi1p;
  kelse p{
    k+ktint nq o= ntransp[npp][ncp];
    kif p(nmxlp[npp] o+ l+m+mi1 o== nmxlp[nqp]) nparp[nnpp] o= nqp;
    kelse p{
      k+ktint nnq o= o++ncntp;
      nmxlp[nnqp] o= nmxlp[npp] o+ l+m+mi1p;
      nmemcpyp(ntransp[nnqp], ntransp[nqp], ksizeofp(ntransp[nnqp]));
      nparp[nnqp] o= nparp[nqp];
      nparp[nnpp] o= nparp[nqp] o= nnqp;
      kfor p(; ntransp[npp][ncp] o== nqp; np o= nparp[npp]) ntransp[npp][ncp] o= nnqp;
    p}
  p}
p}
k+krinline k+ktvoid n+nfbuildsamp() p{
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nlp; o++nip) nextendp(nip, nstrp[nip] o- l+s+sc'a'p);
  nmemsetp(nsump, l+m+mi0p, ksizeofp(o*nsump) o* p(nl o* l+m+mi2 o+ l+m+mi1p));
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= ncntp; o++nip) nsump[nmxlp[nip]]o++p;
  kfor p(k+ktint ni o= l+m+mi1p; ni o<= nlp; o++nip) nsump[nip] o+= nsump[ni o- l+m+mi1p];
  kfor p(k+ktint ni o= ncntp; nip; o--nip) nseqp[nsump[nmxlp[nip]]o--p] o= nip;
  kfor p(k+ktint ni o= ncntp; nip; o--nip) nsizep[nparp[nseqp[nip]]] o+= nsizep[nseqp[nip]];
p}
```

EX 后缀自动机

```
k+krinline k+ktvoid n+nfadd_nodep(k+ktint nxp, k+ktint o&nlastp) p{
  k+ktint nlastnode o= nlastp;
  kif p(ncp[nlastnodep][nxp]) p{
    k+ktint nnownode o= ncp[nlastnodep][nxp];
    kif p(nlp[nnownodep] o== nlp[nlastnodep] o+ l+m+mi1p) nlast o= nnownodep;
    kelse p{
      k+ktint nauxnode o= o++ncntp; nlp[nauxnodep] o= nlp[nlastnodep] o+ l+m+mi1p;
      kfor p(k+ktint ni o= l+m+mi0p; ni o< nalphabetp; o++nip) ncp[nauxnodep][nip] o=
   ↪ ncp[nnownodep][nip];
      nparp[nauxnodep] o= nparp[nnownodep]; nparp[nnownodep] o= nauxnodep;
      kfor p(; nlastnode o&& ncp[nlastnodep][nxp] o== nnownodep; nlastnode o=
   ↪ nparp[nlastnodep]) p{
        ncp[nlastnodep][nxp] o= nauxnodep;
      p}
      nlast o= nauxnodep;
    p}
  p} kelse p{
    k+ktint nnewnode o= o++ncntp; nlp[nnewnodep] o= nlp[nlastnodep] o+ l+m+mi1p;
    kfor p(; nlastnode o&& o!ncp[nlastnodep][nxp]; nlastnode o= nparp[nlastnodep])
   ↪ ncp[nlastnodep][nxp] o= nnewnodep;
    kif p(o!nlastnodep) nparp[nnewnodep] o= l+m+mi1p;
    kelse p{
      k+ktint nnownode o= ncp[nlastnodep][nxp];
      kif p(nlp[nlastnodep] o+ l+m+mi1 o== nlp[nnownodep]) nparp[nnewnodep] o= nnownodep;
      kelse p{
        k+ktint nauxnode o= o++ncntp; nlp[nauxnodep] o= nlp[nlastnodep] o+ l+m+mi1p;
        kfor p(k+ktint ni o= l+m+mi0p; ni o< nalphabetp; o++nip) ncp[nauxnodep][nip] o=
   ↪ ncp[nnownodep][nip];
        nparp[nauxnodep] o= nparp[nnownodep]; nparp[nnownodep] o= nparp[nnewnodep] o=
   ↪ nauxnodep;
        kfor p(; nlastnode o&& ncp[nlastnodep][nxp] o== nnownodep; nlastnode o=
   ↪ nparp[nlastnodep]) p{
          ncp[nlastnodep][nxp] o= nauxnodep;
        p}
      p}
    p}
    nlast o= nnewnodep;
  p}
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

```
k+ktint nnTp, nnStrp, nlastp, ncp[nMAXTp][l+m+mi26p], nfailp[nMAXTp], nrp[nMAXNp],
   ↪ nlp[nMAXNp], nsp[nMAXNp];
k+ktint n+nfallocatep(k+ktint nlenp) p{
  nlp[nnTp] o= nlenp;
  nrp[nnTp] o= l+m+mi0p;
  nfailp[nnTp] o= l+m+mi0p;
  nmemsetp(ncp[nnTp], l+m+mi0p, ksizeofp(ncp[nnTp]));
  kreturn nnTo++p;
p}
k+ktvoid n+nfinitp() p{
  nnT o= nnStr o= l+m+mi0p;
  k+ktint nnewE o= nallocatep(l+m+mi0p);
  k+ktint nnewO o= nallocatep(o-l+m+mi1p);
  nlast o= nnewEp;
  nfailp[nnewEp] o= nnewOp;
  nfailp[nnewOp] o= nnewEp;
  nsp[l+m+mi0p] o= o-l+m+mi1p;
p}
k+ktvoid n+nfaddp(k+ktint nxp) p{
  nsp[o++nnStrp] o= nxp;
  k+ktint nnow o= nlastp;
  kwhile p(nsp[nnStr o- nlp[nnowp] o- l+m+mi1p] o!= nsp[nnStrp]) nnow o= nfailp[nnowp];
  kif p(o!ncp[nnowp][nxp]) p{
    k+ktint nnewnode o= nallocatep(nlp[nnowp] o+ l+m+mi2p), o&nnewfail o= nfailp[nnewnodep];
    nnewfail o= nfailp[nnowp];
    kwhile p(nsp[nnStr o- nlp[nnewfailp] o- l+m+mi1p] o!= nsp[nnStrp]) nnewfail o=
   ↪ nfailp[nnewfailp];
    nnewfail o= ncp[nnewfailp][nxp];
    ncp[nnowp][nxp] o= nnewnodep;
  p}
  nlast o= ncp[nnowp][nxp];
  nrp[nlastp]o++p;
p}
k+ktvoid n+nfcountp() p{
  kfor p(k+ktint ni o= nnT o- l+m+mi1p; ni o>= l+m+mi0p; nio--p) p{
    nrp[nfailp[nip]] o+= nrp[nip];
  p}
p}
```

数据结构

KD-Tree

```
k+ktlong k+ktlong n+nfnormp(kconst k+ktlong k+ktlong o&nxp) p{
    c+c1//    For manhattan distance
    kreturn nstdo::nabsp(nxp);
    c+c1//    For euclid distance
    kreturn nx o* nxp;
p}

kstruct nPoint p{
    k+ktint nxp, nyp, nidp;

    kconst k+ktinto& koperator p[] p(k+ktint nindexp) kconst p{
        kif p(nindex o== l+m+mi0p) p{
            kreturn nxp;
        p} kelse p{
            kreturn nyp;
        p}
    p}

    kfriend k+ktlong k+ktlong ndistp(kconst nPoint o&nap, kconst nPoint o&nbp) p{
        k+ktlong k+ktlong nresult o= l+m+mi0p;
        kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; o++nip) p{
            nresult o+= nnormp(nap[nip] o- nbp[nip]);
        p}
        kreturn nresultp;
    p}
p} npointp[nNp];

kstruct nRectangle p{
    k+ktint nminp[l+m+mi2p], nmaxp[l+m+mi2p];

    nRectanglep() p{
        nminp[l+m+mi0p] o= nminp[l+m+mi1p] o= nINT_MAXp;  c+c1// sometimes int is not enough
        nmaxp[l+m+mi0p] o= nmaxp[l+m+mi1p] o= nINT_MINp;
    p}

    k+ktvoid naddp(kconst nPoint o&npp) p{
        kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; o++nip) p{
            nminp[nip] o= nstdo::nminp(nminp[nip], npp[nip]);
            nmaxp[nip] o= nstdo::nmaxp(nmaxp[nip], npp[nip]);
        p}
    p}

    k+ktlong k+ktlong ndistp(kconst nPoint o&npp) p{
        k+ktlong k+ktlong nresult o= l+m+mi0p;
        kfor p(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; o++nip) p{
            c+c1//    For minimum distance
            nresult o+= nnormp(nstdo::nminp(nstdo::nmaxp(npp[nip], nminp[nip]), nmaxp[nip])
   ↪ o- npp[nip]);
            c+c1//    For maximum distance
            nresult o+= nstdo::nmaxp(nnormp(nmaxp[nip] o- npp[nip]), nnormp(nminp[nip] o-
   ↪ npp[nip]));
        p}
        kreturn nresultp;
    p}
p};

kstruct nNode p{
    nPoint nseperatorp;
    nRectangle nrectanglep;
    k+ktint nchildp[l+m+mi2p];

    k+ktvoid n+nfresetp(kconst nPoint o&npp) p{
        nseperator o= npp;
        nrectangle o= nRectanglep();
```

```
        nrectanglep.naddp(npp);
        nchildp[l+m+mi0p] o= nchildp[l+m+mi1p] o= l+m+mi0p;
    p}
p} ntreep[nN o<< l+m+mi1p];

k+ktint nsizep, npivotp;

k+ktbool n+nfcomparep(kconst nPoint o&nap, kconst nPoint o&nbp) p{
    kif p(nap[npivotp] o!= nbp[npivotp]) p{
        kreturn nap[npivotp] o< nbp[npivotp];
    p}
    kreturn nap.nid o< nbp.nidp;
p}

c+c1// 左閉右開: build(1, n + 1)
k+ktint n+nfbuildp(k+ktint nlp, k+ktint nrp, k+ktint ntype o= l+m+mi1p) p{
    npivot o= ntypep;
    kif p(nl o>= nrp) p{
        kreturn l+m+mi0p;
    p}
    k+ktint nx o= o++nsizep;
    k+ktint nmid o= nl o+ nr o>> l+m+mi1p;
    nstdo::nnth_elementp(npoint o+ nlp, npoint o+ nmidp, npoint o+ nrp, ncomparep);
    ntreep[nxp].nresetp(npointp[nmidp]);
    kfor p(k+ktint ni o= nlp; ni o< nrp; o++nip) p{
        ntreep[nxp].nrectanglep.naddp(npointp[nip]);
    p}
    ntreep[nxp].nchildp[l+m+mi0p] o= nbuildp(nlp, nmidp, ntype o^ l+m+mi1p);
    ntreep[nxp].nchildp[l+m+mi1p] o= nbuildp(nmid o+ l+m+mi1p, nrp, ntype o^ l+m+mi1p);
    kreturn nxp;
p}

k+ktint n+nfinsertp(k+ktint nxp, kconst nPoint o&npp, k+ktint ntype o= l+m+mi1p) p{
    npivot o= ntypep;
    kif p(nx o== l+m+mi0p) p{
        ntreep[o++nsizep].nresetp(npp);
        kreturn nsizep;
    p}
    ntreep[nxp].nrectanglep.naddp(npp);
    kif p(ncomparep(npp, ntreep[nxp].nseperatorp)) p{
        ntreep[nxp].nchildp[l+m+mi0p] o= ninsertp(ntreep[nxp].nchildp[l+m+mi0p], npp, ntype
   ↪ o^ l+m+mi1p);
    p} kelse p{
        ntreep[nxp].nchildp[l+m+mi1p] o= ninsertp(ntreep[nxp].nchildp[l+m+mi1p], npp, ntype
   ↪ o^ l+m+mi1p);
    p}
    kreturn nxp;
p}

c+c1// For minimum distance
c+c1// For maximum: 下面递归 query 时 0, 1 换顺序;< and >;min and max
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{
    npivot o= ntypep;
    kif p(nx o== l+m+mi0 o|| ntreep[nxp].nrectanglep.ndistp(npp) o> nanswerp.nfirstp) p{
        kreturnp;
    p}
    nanswer o= nstdo::nminp(nanswerp,
            nstdo::nmake_pairp(ndistp(ntreep[nxp].nseperatorp, npp),
   ↪ ntreep[nxp].nseperatorp.nidp));
    kif p(ncomparep(npp, ntreep[nxp].nseperatorp)) p{
        nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nanswerp, ntype o^ l+m+mi1p);
        nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nanswerp, ntype o^ l+m+mi1p);
    p} kelse p{
        nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nanswerp, ntype o^ l+m+mi1p);
        nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nanswerp, ntype o^ l+m+mi1p);
    p}
p}

nstdo::npriority_queueo<nstdo::npairo<k+ktlong k+ktlongp, k+ktinto> o> nanswerp;

k+ktvoid n+nfqueryp(k+ktint nxp, kconst nPoint o&npp, k+ktint nkp, k+ktint ntype o= l+m+mi1p)
   ↪ p{
    npivot o= ntypep;
    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{
        kreturnp;
    p}
    nanswerp.npushp(nstdo::nmake_pairp(ndistp(ntreep[nxp].nseperatorp, npp),
   ↪ ntreep[nxp].nseperatorp.nidp));
    kif p((k+ktintp)nanswerp.nsizep() o> nkp) p{
        nanswerp.npopp();
    p}
    kif p(ncomparep(npp, ntreep[nxp].nseperatorp)) p{
        nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nkp, ntype o^ l+m+mi1p);
        nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nkp, ntype o^ l+m+mi1p);
    p} kelse p{
        nqueryp(ntreep[nxp].nchildp[l+m+mi1p], npp, nkp, ntype o^ l+m+mi1p);
        nqueryp(ntreep[nxp].nchildp[l+m+mi0p], npp, nkp, ntype o^ l+m+mi1p);
    p}
p}
```

Treap

```
kstruct nNodep{
  k+ktint nmnp, nkeyp, nsizep, ntagp;
  k+ktbool nrevp;
  nNodeo* nchp[l+m+mi2p];
  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){}
  k+ktvoid ndowntagp();
  nNodeo* n+nfupdatep(){
    nmn o= nminp(nchp[l+m+mi0p] o-> nmnp, nminp(nkeyp, nchp[l+m+mi1p] o-> nmnp));
    nsize o= nchp[l+m+mi0p] o-> nsize o+ l+m+mi1 o+ nchp[l+m+mi1p] o-> nsizep;
    kreturn kthisp;
  p}
p};
ktypedef npairo<nNodeo*p, nNodeo*> nPairp;
nNode o*nnullp, o*nrootp;
k+ktvoid nNodeo::ndowntagp(){
```

```
  kifp(nrevp){
    kforp(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; nio++p)
      kifp(nchp[nip] o!= nnullp){
        nchp[nip] o-> nrev o^= l+m+mi1p;
        nswapp(nchp[nip] o-> nchp[l+m+mi0p], nchp[nip] o-> nchp[l+m+mi1p]);
      p}
    nrev o= l+m+mi0p;
  p}
  kifp(ntagp){
    kforp(k+ktint ni o= l+m+mi0p; ni o< l+m+mi2p; nio++p)
      kifp(nchp[nip] o!= nnullp){
        nchp[nip] o-> nkey o+= ntagp;
        nchp[nip] o-> nmn o+= ntagp;
        nchp[nip] o-> ntag o+= ntagp;
      p}
    ntag o= l+m+mi0p;
  p}
p}
k+ktint nrp(){
  kstatic k+ktint ns o= l+m+mi3023192386p;
  kreturn p(ns o+= p(ns o<< l+m+mi3p) o+ l+m+mi1p) o& p(o~l+m+mi0u o>> l+m+mi1p);
p}
k+ktbool nrandomp(k+ktint nxp, k+ktint nyp){
  kreturn nrp() o% p(nx o+ nyp) o< nxp;
p}
nNodeo* nmergep(nNode o*npp, nNode o*nqp){
  kifp(np o== nnullp) kreturn nqp;
  kifp(nq o== nnullp) kreturn npp;
  np o-> ndowntagp();
  nq o-> ndowntagp();
  kifp(nrandomp(np o-> nsizep, nq o-> nsizep)){
    np o-> nchp[l+m+mi1p] o= nmergep(np o-> nchp[l+m+mi1p], nqp);
    kreturn np o-> nupdatep();
  p}kelsep{
    nq o-> nchp[l+m+mi0p] o= nmergep(npp, nq o-> nchp[l+m+mi0p]);
    kreturn nq o-> nupdatep();
  p}
p}
nPair nsplitp(nNode o*nxp, k+ktint nnp){
  kifp(nx o== nnullp) kreturn nmake_pairp(nnullp, nnullp);
  nx o-> ndowntagp();
  kifp(nn o<= nx o-> nchp[l+m+mi0p] o-> nsizep){
    nPair nret o= nsplitp(nx o-> nchp[l+m+mi0p], nnp);
    nx o-> nchp[l+m+mi0p] o= nretp.nsecondp;
    kreturn n+nfmake_pairp(nretp.nfirstp, nx o-> nupdatep());
  p}
  nPair nret o= nsplitp(nx o-> nchp[l+m+mi1p], nn o- nx o-> nchp[l+m+mi0p] o-> nsize o-
   ↪ l+m+mi1p);
  nx o-> nchp[l+m+mi1p] o= nretp.nfirstp;
  kreturn n+nfmake_pairp(nx o-> nupdatep(), nretp.nsecondp);
p}
npairo<nNodeo*p, nPairo> nget_segmentp(k+ktint nlp, k+ktint nrp){
  nPair nret o= nsplitp(nrootp, nl o- l+m+mi1p);
  kreturn n+nfmake_pairp(nretp.nfirstp, nsplitp(nretp.nsecondp, nr o- nl o+ l+m+mi1p));
p}
```

```
k+ktint nmainp(){
  nnull o= knew nNodep(nINFp, nINFp, l+m+mi0p);
  nnull o-> nchp[l+m+mi0p] o= nnull o-> nchp[l+m+mi1p] o= nnullp;
  nroot o= nnullp;
p}
```

Link/cut Tree

```
k+krinline k+ktvoid n+nfreversep(k+ktint nxp) p{
  ntrp[nxp].nrev o^= l+m+mi1p; nswapp(ntrp[nxp].ncp[l+m+mi0p], ntrp[nxp].ncp[l+m+mi1p]);
p}

k+krinline k+ktvoid n+nfrotatep(k+ktint nxp, k+ktint nkp) p{
  k+ktint ny o= ntrp[nxp].nfap, nz o= ntrp[nyp].nfap;
    ntrp[nxp].nfa o= nzp; ntrp[nzp].ncp[ntrp[nzp].ncp[l+m+mi1p] o== nyp] o= nxp;
    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];
    ntrp[nxp].ncp[nk o^ l+m+mi1p] o= nyp; ntrp[nyp].nfa o= nxp;
p}

k+krinline k+ktvoid n+nfsplayp(k+ktint nxp, k+ktint nwp) p{
  k+ktint nz o= nxp; npushdownp(nxp);
  kwhile p(ntrp[nxp].nfa o!= nwp) p{
    k+ktint ny o= ntrp[nxp].nfap; nz o= ntrp[nyp].nfap;
    kif p(nz o== nwp) p{
      npushdownp(nz o= nyp); npushdownp(nxp);
      nrotatep(nxp, ntrp[nyp].ncp[l+m+mi1p] o== nxp);
      nupdatep(nyp); nupdatep(nxp);
    p} kelse p{
      npushdownp(nzp); npushdownp(nyp); npushdownp(nxp);
      k+ktint nt1 o= ntrp[nyp].ncp[l+m+mi1p] o== nxp, nt2 o= ntrp[nzp].ncp[l+m+mi1p] o== nyp;
      kif p(nt1 o== nt2p) nrotatep(nyp, nt2p), nrotatep(nxp, nt1p);
      kelse nrotatep(nxp, nt1p), nrotatep(nxp, nt2p);
      nupdatep(nzp); nupdatep(nyp); nupdatep(nxp);
    p}
  p}
  nupdatep(nxp);
  kif p(nx o!= nzp) nparp[nxp] o= nparp[nzp], nparp[nzp] o= l+m+mi0p;
p}

k+krinline k+ktvoid n+nfaccessp(k+ktint nxp) p{
  kfor p(k+ktint ny o= l+m+mi0p; nxp; ny o= nxp, nx o= nparp[nxp]) p{
    nsplayp(nxp, l+m+mi0p);
    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;
    ntrp[nxp].ncp[l+m+mi1p] o= nyp; nparp[nyp] o= l+m+mi0p; ntrp[nyp].nfa o= nxp;
   ↪ nupdatep(nxp);
  p}
p}

k+krinline k+ktvoid n+nfmakerootp(k+ktint nxp) p{
  naccessp(nxp); nsplayp(nxp, l+m+mi0p); nreversep(nxp);
p}

k+krinline k+ktvoid n+nflinkp(k+ktint nxp, k+ktint nyp) p{
  nmakerootp(nxp); nparp[nxp] o= nyp;
```

```
p}

k+krinline k+ktvoid n+nfcutp(k+ktint nxp, k+ktint nyp) p{
  naccessp(nxp); nsplayp(nyp, l+m+mi0p);
  kif p(nparp[nyp] o!= nxp) nswapp(nxp, nyp), naccessp(nxp), nsplayp(nyp, l+m+mi0p);
  nparp[nyp] o= l+m+mi0p;
p}

k+krinline k+ktvoid n+nfsplitp(k+ktint nxp, k+ktint nyp) p{  c+c1// x will be the root of the tree
  nmakerootp(nyp); naccessp(nxp); nsplayp(nxp, l+m+mi0p);
p}
```

树状数组查询第 k 小元素

```
k+krinline k+ktvoid n+nfreversep(k+ktint nxp) p{
  ntrp[nxp].nrev o^= l+m+mi1p; nswapp(ntrp[nxp].ncp[l+m+mi0p], ntrp[nxp].ncp[l+m+mi1p]);
p}

k+krinline k+ktvoid n+nfrotatep(k+ktint nxp, k+ktint nkp) p{
  k+ktint ny o= ntrp[nxp].nfap, nz o= ntrp[nyp].nfap;
    ntrp[nxp].nfa o= nzp; ntrp[nzp].ncp[ntrp[nzp].ncp[l+m+mi1p] o== nyp] o= nxp;
    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];
    ntrp[nxp].ncp[nk o^ l+m+mi1p] o= nyp; ntrp[nyp].nfa o= nxp;
p}

k+krinline k+ktvoid n+nfsplayp(k+ktint nxp, k+ktint nwp) p{
  k+ktint nz o= nxp; npushdownp(nxp);
  kwhile p(ntrp[nxp].nfa o!= nwp) p{
    k+ktint ny o= ntrp[nxp].nfap; nz o= ntrp[nyp].nfap;
    kif p(nz o== nwp) p{
      npushdownp(nz o= nyp); npushdownp(nxp);
      nrotatep(nxp, ntrp[nyp].ncp[l+m+mi1p] o== nxp);
      nupdatep(nyp); nupdatep(nxp);
    p} kelse p{
      npushdownp(nzp); npushdownp(nyp); npushdownp(nxp);
      k+ktint nt1 o= ntrp[nyp].ncp[l+m+mi1p] o== nxp, nt2 o= ntrp[nzp].ncp[l+m+mi1p] o== nyp;
      kif p(nt1 o== nt2p) nrotatep(nyp, nt2p), nrotatep(nxp, nt1p);
      kelse nrotatep(nxp, nt1p), nrotatep(nxp, nt2p);
      nupdatep(nzp); nupdatep(nyp); nupdatep(nxp);
    p}
  p}
  nupdatep(nxp);
  kif p(nx o!= nzp) nparp[nxp] o= nparp[nzp], nparp[nzp] o= l+m+mi0p;
p}

k+krinline k+ktvoid n+nfaccessp(k+ktint nxp) p{
  kfor p(k+ktint ny o= l+m+mi0p; nxp; ny o= nxp, nx o= nparp[nxp]) p{
    nsplayp(nxp, l+m+mi0p);
    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;
    ntrp[nxp].ncp[l+m+mi1p] o= nyp; nparp[nyp] o= l+m+mi0p; ntrp[nyp].nfa o= nxp;
    nupdatep(nxp);
  p}
p}
```

```

k+krinline k+ktvoid n+nfmakerootp(k+ktint nxp) p{
  naccessp(nxp); nsplayp(nxp, l+m+mi0p); nreversep(nxp);
p}

k+krinline k+ktvoid n+nflinkp(k+ktint nxp, k+ktint nyp) p{
  nmakerootp(nxp); nparp[nxp] o= nyp;
p}

k+krinline k+ktvoid n+nfcutp(k+ktint nxp, k+ktint nyp) p{
  naccessp(nxp); nsplayp(nyp, l+m+mi0p);
  kif p(nparp[nyp] o!= nxp) nswapp(nxp, nyp), naccessp(nxp), nsplayp(nyp, l+m+mi0p);
  nparp[nyp] o= l+m+mi0p;
p}

k+krinline k+ktvoid n+nfsplitp(k+ktint nxp, k+ktint nyp) p{  c+c1// x will be the root of the tree
  nmakerootp(nyp); naccessp(nxp); nsplayp(nxp, l+m+mi0p);
p}
```

图论

基础

```
kstruct nGraph p{  c+c1// Remember to call .init()!
    k+ktint nep, nnxtp[nMp], nvp[nMp], nadjp[nNp], nnp;
    k+ktbool nbasep;
    k+kr__inline k+ktvoid n+nfinitp(k+ktbool n_basep, k+ktint n_n o= l+m+mi0p) p{
        nassertp(nn o< nNp);
        nn o= n_np; nbase o= n_basep;
        ne o= l+m+mi0p; nmemsetp(nadj o+ nbasep, o-l+m+mi1p, ksizeofp(o*nadjp) o* nnp);
    p}
    k+kr__inline k+ktint n+nfnew_nodep() p{
        nadjp[nn o+ nbasep] o= o-l+m+mi1p;
        nassertp(nn o+ nbase o+ l+m+mi1 o< nNp);
        kreturn nno++ o+ nbasep;
    p}
    k+kr__inline k+ktvoid n+nfinsp(k+ktint nu0p, k+ktint nv0p) p{  c+c1// directional
        nassertp(nu0 o< nn o+ nbase o&& nv0 o< nn o+ nbasep);
        nvp[nep] o= nv0p; nnxtp[nep] o= nadjp[nu0p]; nadjp[nu0p] o= neo++p;
        nassertp(ne o< nMp);
    p}
    k+kr__inline k+ktvoid n+nfbi_insp(k+ktint nu0p, k+ktint nv0p) p{  c+c1// bi-directional
        ninsp(nu0p, nv0p); ninsp(nv0p, nu0p);
    p}
p};
```

KM

```
kstruct nKM p{
  c+c1// Truly O(n^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 -INF，使用时先对 1 -> n 调用 hungary() ，再 get_ans() 求值
  k+ktint nwp[nNp][nNp];
  k+ktint nlxp[nNp], nlyp[nNp], nmatchp[nNp], nwayp[nNp], nslackp[nNp];
```

```
  k+ktbool nusedp[nNp];
  k+ktvoid n+nfinitp() p{
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nnp; nio++p) p{
      nmatchp[nip] o= l+m+mi0p;
      nlxp[nip] o= l+m+mi0p;
      nlyp[nip] o= l+m+mi0p;
      nwayp[nip] o= l+m+mi0p;
    p}
  p}
  k+ktvoid n+nfhungaryp(k+ktint nxp) p{
    nmatchp[l+m+mi0p] o= nxp;
    k+ktint nj0 o= l+m+mi0p;
    kfor p(k+ktint nj o= l+m+mi0p; nj o<= nnp; njo++p) p{
      nslackp[njp] o= nINFp;
      nusedp[njp] o= n+nbfalsep;
    p}

    kdo p{
      nusedp[nj0p] o= n+nbtruep;
      k+ktint ni0 o= nmatchp[nj0p], ndelta o= nINFp, nj1 o= l+m+mi0p;
      kfor p(k+ktint nj o= l+m+mi1p; nj o<= nnp; njo++p) p{
        kif p(nusedp[njp] o== n+nbfalsep) p{
          k+ktint ncur o= o-nwp[ni0p][njp] o- nlxp[ni0p] o- nlyp[njp];
          kif p(ncur o< nslackp[njp]) p{
            nslackp[njp] o= ncurp;
            nwayp[njp] o= nj0p;
          p}
          kif p(nslackp[njp] o< ndeltap) p{
            ndelta o= nslackp[njp];
            nj1 o= njp;
          p}
        p}
      p}
      kfor p(k+ktint nj o= l+m+mi0p; nj o<= nnp; njo++p) p{
        kif p(nusedp[njp]) p{
          nlxp[nmatchp[njp]] o+= ndeltap;
          nlyp[njp] o-= ndeltap;
        p}
        kelse nslackp[njp] o-= ndeltap;
      p}
      nj0 o= nj1p;
    p} kwhile p(nmatchp[nj0p] o!= l+m+mi0p);

    kdo p{
      k+ktint nj1 o= nwayp[nj0p];
      nmatchp[nj0p] o= nmatchp[nj1p];
      nj0 o= nj1p;
    p} kwhile p(nj0p);
  p}

  k+ktint n+nfget_ansp() p{
    k+ktint nsum o= l+m+mi0p;
    kforp(k+ktint ni o= l+m+mi1p; ni o<= nnp; nio++p) p{
      kif p(nwp[nmatchp[nip]][nip] o== o-nINFp) p; c+c1// 无解
      kif p(nmatchp[nip] o> l+m+mi0p) nsum o+= nwp[nmatchp[nip]][nip];
    p}
    kreturn nsump;
  p}
p} nkmp;
```

点双连通分量

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

```
kconst k+ktbool nBCC_VERTEX o= l+m+mi0p, nBCC_EDGE o= l+m+mi1p;
kstruct nBCC p{  c+c1// N = N0 + M0. Remember to call init(&raw_graph).
  nGraph o*ngp, nforestp; c+c1// g is raw graph ptr.
  k+ktint ndfnp[nNp], nDFNp, nlowp[nNp];
  k+ktint nstackp[nNp], ntopp;
  k+ktint nexpand_top[nNp];    c+c1// Where edge i is expanded to in expaned graph.
  c+c1// Vertex i expaned to i.
  k+ktint ncompress_top[nNp];  c+c1// Where vertex i is compressed to.
  k+ktbool nvertex_typep[nNp], ncutp[nNp], ncompress_cutp[nNp], nbranchp[nMp];
  c+c1//std::vector<int> BCC_component[N];  // Cut vertex belongs to none.
  k+kr__inline k+ktvoid n+nfinitp(nGraph o*nraw_graphp) p{
    ng o= nraw_graphp;
  p}
  k+ktvoid n+nfDFSp(k+ktint nup, k+ktint npep) p{
    ndfnp[nup] o= nlowp[nup] o= o++nDFNp; ncutp[nup] o= n+nbfalsep;
    kif p(o!~ngo->nadjp[nup]) p{
      ncutp[nup] o= l+m+mi1p;
      ncompress_top[nup] o= nforestp.nnew_nodep();
      ncompress_cutp[ncompress_top[nup]] o= l+m+mi1p;
    p}
    kfor p(k+ktint ne o= ngo->nadjp[nup]; o~nep; ne o= ngo->nnxtp[nep]) p{
      k+ktint nv o= ngo->nvp[nep];
      kif p((ne o^ npep) o> l+m+mi1 o&& ndfnp[nvp] o> l+m+mi0 o&& ndfnp[nvp] o< ndfnp[nup])
  ↪ p{
        nstackp[ntopo++p] o= nep;
        nlowp[nup] o= nstdo::nminp(nlowp[nup], ndfnp[nvp]);
      p}
      kelse kif p(o!ndfnp[nvp]) p{
        nstackp[ntopo++p] o= nep; nbranchp[nep] o= l+m+mi1p;
        nDFSp(nvp, nep);
        nlowp[nup] o= nstdo::nminp(nlowp[nvp], nlowp[nup]);
        kif p(nlowp[nvp] o>= ndfnp[nup]) p{
          kif p(o!ncutp[nup]) p{
            ncutp[nup] o= l+m+mi1p;
            ncompress_top[nup] o= nforestp.nnew_nodep();
            ncompress_cutp[ncompress_top[nup]] o= l+m+mi1p;
          p}
          k+ktint ncc o= nforestp.nnew_nodep();
          nforestp.nbi_insp(ncompress_top[nup], nccp);
          ncompress_cutp[nccp] o= l+m+mi0p;
          c+c1//BCC_component[cc].clear();
          kdo p{
            k+ktint ncur_e o= nstackp[o--ntopp];
            ncompress_top[nexpand_top[ncur_ep]] o= nccp;
            ncompress_top[nexpand_top[ncur_eo^l+m+mi1p]] o= nccp;
            kif p(nbranchp[ncur_ep]) p{
              k+ktint nv o= ngo->nvp[ncur_ep];
```

```
              kif p(ncutp[nvp])
                nforestp.nbi_insp(nccp, ncompress_top[nvp]);
              kelse p{
                c+c1//BCC_component[cc].push_back(v);
                ncompress_top[nvp] o= nccp;
              p}
            p}
          p} kwhile p(nstackp[ntopp] o!= nep);
        p}
      p}
    p}
  p}
  k+ktvoid n+nfsolvep() p{
    nforestp.ninitp(ngo->nbasep);
    k+ktint nn o= ngo->nnp;
    kfor p(k+ktint ni o= l+m+mi0p; ni o< ngo->nep; nio++p) p{
      nexpand_top[nip] o= ngo->nnew_nodep();
    p}
    nmemsetp(nbranchp, l+m+mi0p, ksizeofp(o*nbranchp) o* ngo->nep);
    nmemsetp(ndfn o+ ngo->nbasep, l+m+mi0p, ksizeofp(o*ndfnp) o* nnp); nDFN o= l+m+mi0p;
    kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; nio++p)
      kif p(o!ndfnp[ni o+ ngo->nbasep]) p{
        ntop o= l+m+mi0p;
        nDFSp(ni o+ ngo->nbasep, o-l+m+mi1p);
      p}
  p}
p} nbccp;

nbccp.ninitp(o&nraw_graphp);
nbccp.nsolvep();
c+c1// Do something with bcc.forest ...
```

边双连通分量

```
kstruct nBCC p{
  nGraph o*ngp, nforestp;
  k+ktint ndfnp[nNp], nlowp[nNp], nstackp[nNp], ntotp[nNp], nbelongp[nNp], nvisp[nNp], ntopp,
   ↪ ndfs_clockp;
  c+c1// tot[] is the size of each BCC, belong[] is the BCC that each node belongs to
  npairo<k+ktintp, k+ktint o> norip[nMp]; c+c1// bridge in raw_graph(raw node)
  k+ktbool nis_bridgep[nMp];
  k+kr__inline k+ktvoid n+nfinitp(nGraph o*nraw_graphp) p{
    ng o= nraw_graphp;
    nmemsetp(nis_bridgep, n+nbfalsep, ksizeofp(o*nis_bridgep) o* ng o-> nep);
    nmemsetp(nvis o+ ng o-> nbasep, l+m+mi0p, ksizeofp(o*nvisp) o* ng o-> nnp);
  p}
  k+ktvoid n+nftarjanp(k+ktint nup, k+ktint nfromp) p{
    ndfnp[nup] o= nlowp[nup] o= o++ndfs_clockp; nvisp[nup] o= l+m+mi1p; nstackp[o++ntopp] o=
   ↪ nup;
    kfor p(k+ktint np o= ng o-> nadjp[nup]; o~npp; np o= ng o-> nnxtp[npp]) p{
      kif p((np o^ l+m+mi1p) o== nfromp) kcontinuep;
      k+ktint nv o= ng o-> nvp[npp];
      kif p(nvisp[nvp]) p{
        kif p(nvisp[nvp] o== l+m+mi1p) nlowp[nup] o= nminp(nlowp[nup], ndfnp[nvp]);
      p} kelse p{
        ntarjanp(nvp, npp);
        nlowp[nup] o= nminp(nlowp[nup], nlowp[nvp]);
        kif p(nlowp[nvp] o> ndfnp[nup]) nis_bridgep[np o/ l+m+mi2p] o= n+nbtruep;
      p}
    p}
    kif p(ndfnp[nup] o!= nlowp[nup]) kreturnp;
    ntotp[nforestp.nnew_nodep()] o= l+m+mi0p;
    kdo p{
      nbelongp[nstackp[ntopp]] o= nforestp.nnp;
      nvisp[nstackp[ntopp]] o= l+m+mi2p;
      ntotp[nforestp.nnp]o++p;
      o--ntopp;
    p} kwhile p(nstackp[ntop o+ l+m+mi1p] o!= nup);
  p}
  k+ktvoid n+nfsolvep() p{
    nforestp.ninitp(ng o-> nbasep);
    k+ktint nn o= ng o-> nnp;
    kfor p(k+ktint ni o= l+m+mi0p; ni o< nnp; o++nip)
      kif p(o!nvisp[ni o+ ng o-> nbasep]) p{
        ntop o= ndfs_clock o= l+m+mi0p;
        ntarjanp(ni o+ ng o-> nbasep, o-l+m+mi1p);
      p}
    kfor p(k+ktint ni o= l+m+mi0p; ni o< ng o-> ne o/ l+m+mi2p; o++nip)
      kif p(nis_bridgep[nip]) p{
        k+ktint ne o= nforestp.nep;
        nforestp.nbi_insp(nbelongp[ng o-> nvp[ni o* l+m+mi2p]], nbelongp[ng o-> nvp[ni o*
   ↪ l+m+mi2 o+ l+m+mi1p]], ng o-> nwp[ni o* l+m+mi2p]);
        norip[nep] o= nmake_pairp(ng o-> nvp[ni o* l+m+mi2 o+ l+m+mi1p], ng o-> nvp[ni o*
   ↪ l+m+mi2p]);
        norip[ne o+ l+m+mi1p] o= nmake_pairp(ng o-> nvp[ni o* l+m+mi2p], ng o-> nvp[ni o*
   ↪ l+m+mi2 o+ l+m+mi1p]);
      p}
  p}
p} nbccp;
```

最小树形图

```
kconst k+ktint nMAXNp,nINFp;c+c1// INF >= sum( W_ij )
k+ktint nfromp[nMAXN o+ l+m+mi10p][nMAXN o* l+m+mi2 o+ l+m+mi10p],nnp,nmp,nedgep[nMAXN o+
   ↪ l+m+mi10p][nMAXN o* l+m+mi2 o+ l+m+mi10p];
k+ktint nselp[nMAXN o* l+m+mi2 o+ l+m+mi10p],nfap[nMAXN o* l+m+mi2 o+ l+m+mi10p],nvisp[nMAXN
   ↪ o* l+m+mi2 o+ l+m+mi10p];
k+ktint n+nfgetfap(k+ktint nxp){kifp(nx o== nfap[nxp]) kreturn nxp; kreturn nfap[nxp] o=
   ↪ ngetfap(nfap[nxp]);}
k+ktvoid n+nfliuzhup(){ c+c1// 1-base: root is 1, answer = (sel[i], i) for i in [2..n]
  nfap[l+m+mi1p] o= l+m+mi1p;
  kforp(k+ktint ni o= l+m+mi2p; ni o<= nnp; o++nip){
    nselp[nip] o= l+m+mi1p; nfap[nip] o= nip;
    kforp(k+ktint nj o= l+m+mi1p; nj o<= nnp; o++njp) kifp(nfap[njp] o!= nip)
      kifp(nfromp[njp][nip] o= nip, nedgep[nselp[nip]][nip] o> nedgep[njp][nip]) nselp[nip]
   ↪ o= njp;
  p}
  k+ktint nlimit o= nnp;
  kwhilep(l+m+mi1p){
    k+ktint nprelimit o= nlimitp; nmemsetp(nvisp, l+m+mi0p, ksizeofp(nvisp)); nvisp[l+m+mi1p]
   ↪ o= l+m+mi1p;
```

```
    kforp(k+ktint ni o= l+m+mi2p; ni o<= nprelimitp; o++nip) kifp(nfap[nip] o== ni o&&
    ↪ o!nvisp[nip]){
      k+ktint nj o= nip; kwhilep(o!nvisp[njp]) nvisp[njp] o= nip, nj o= ngetfap(nselp[njp]);
      kifp(nj o== l+m+mi1 o|| nvisp[njp] o!= nip) kcontinuep; nvectoro<k+ktinto> nCp; k+ktint
    ↪ nk o= njp;
      kdo nCp.npush_backp(nkp), nk o= ngetfap(nselp[nkp]); kwhilep(nk o!= njp);
      o++nlimitp;
      kforp(k+ktint ni o= l+m+mi1p; ni o<= nnp; o++nip){
        nedgep[nip][nlimitp] o= nINFp, nfromp[nip][nlimitp] o= nlimitp;
      p}
      nfap[nlimitp] o= nvisp[nlimitp] o= nlimitp;
      kforp(k+ktint ni o= l+m+mi0p; ni o< k+ktintp(nCp.nsizep()); o++nip){
        k+ktint nx o= nCp[nip], nfap[nxp] o= nlimitp;
        kforp(k+ktint nj o= l+m+mi1p; nj o<= nnp; o++njp)
          kifp(nedgep[njp][nxp] o!= nINF o&& nedgep[njp][nlimitp] o> nedgep[njp][nxp] o-
    ↪ nedgep[nselp[nxp]][nxp]){
            nedgep[njp][nlimitp] o= nedgep[njp][nxp] o- nedgep[nselp[nxp]][nxp];
            nfromp[njp][nlimitp] o= nxp;
          p}
      p}
      kforp(k+ktint njo=l+m+mi1p;njo<=nnp;o++njp) kifp(ngetfap(njp)o==nlimitp)
    ↪ nedgep[njp][nlimitp] o= nINFp;
      nselp[nlimitp] o= l+m+mi1p;
      kforp(k+ktint nj o= l+m+mi1p; nj o<= nnp; o++njp)
        kifp(nedgep[nselp[nlimitp]][nlimitp] o> nedgep[njp][nlimitp]) nselp[nlimitp] o= njp;
    p}
    kifp(nprelimit o== nlimitp) kbreakp;
  p}
  kforp(k+ktint ni o= nlimitp; ni o> l+m+mi1p; o--nip) nselp[nfromp[nselp[nip]][nip]] o=
    ↪ nselp[nip];
p}
```

带花树

```
nvectoro<k+ktinto> nlinkp[nmaxnp];
k+ktint nnp,nmatchp[nmaxnp],nQueuep[nmaxnp],nheadp,ntailp;
k+ktint npredp[nmaxnp],nbasep[nmaxnp],nstartp,nfinishp,nnewbasep;
k+ktbool nInQueuep[nmaxnp],nInBlossomp[nmaxnp];
k+ktvoid n+nfpushp(k+ktint nup){ nQueuep[ntailo++p]o=nup;nInQueuep[nup]o=n+nbtruep; p}
k+ktint n+nfpopp(){ kreturn nQueuep[nheado++p]; p}
k+ktint n+nfFindCommonAncestorp(k+ktint nup,k+ktint nvp){
  k+ktbool nInPathp[nmaxnp];
  kforp(k+ktint nio=l+m+mi0p;nio<nnp;nio++p) nInPathp[nip]o=l+m+mi0p;
  kwhilep(n+nbtruep){ nuo=nbasep[nup];nInPathp[nup]o=n+nbtruep;kifp(nuo==nstartp)
    ↪ kbreakp;nuo=npredp[nmatchp[nup]]; p}
  kwhilep(n+nbtruep){ nvo=nbasep[nvp];kifp(nInPathp[nvp]) kbreakp;nvo=npredp[nmatchp[nvp]];
    ↪ p}
  kreturn nvp;
p}
k+ktvoid n+nfResetTracep(k+ktint nup){
  k+ktint nvp;
  kwhilep(nbasep[nup]o!=nnewbasep){
    nvo=nmatchp[nup];
    nInBlossomp[nbasep[nup]]o=nInBlossomp[nbasep[nvp]]o=n+nbtruep;
    nuo=npredp[nvp];
    kifp(nbasep[nup]o!=nnewbasep) npredp[nup]o=nvp;
  p}
p}
k+ktvoid n+nfBlossomContractp(k+ktint nup,k+ktint nvp){
  nnewbaseo=nFindCommonAncestorp(nup,nvp);
  kfor p(k+ktint nio=l+m+mi0p;nio<nnp;nio++p)
  nInBlossomp[nip]o=l+m+mi0p;
  nResetTracep(nup);nResetTracep(nvp);
  kifp(nbasep[nup]o!=nnewbasep) npredp[nup]o=nvp;
  kifp(nbasep[nvp]o!=nnewbasep) npredp[nvp]o=nup;
  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip)
  kifp(nInBlossomp[nbasep[nip]]){
    nbasep[nip]o=nnewbasep;
    kifp(o!nInQueuep[nip]) npushp(nip);
  p}
p}
k+ktbool n+nfFindAugmentingPathp(k+ktint nup){
  k+ktbool nfoundo=n+nbfalsep;
  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip) npredp[nip]o=-l+m+mi1p,nbasep[nip]o=nip;
  kfor p(k+ktint nio=l+m+mi0p;nio<nnp;nio++p) nInQueuep[nip]o=l+m+mi0p;
  nstarto=nup;nfinisho=-l+m+mi1p; nheado=ntailo=l+m+mi0p; npushp(nstartp);
  kwhilep(nheado<ntailp){
    k+ktint nuo=npopp();
    kforp(k+ktint nio=nlinkp[nup].nsizep()o-l+m+mi1p;nio>=l+m+mi0p;nio--p){
      k+ktint nvo=nlinkp[nup][nip];
      kifp(nbasep[nup]o!=nbasep[nvp]o&&nmatchp[nup]o!=nvp)
        kifp(nvo==nstarto||p(nmatchp[nvp]o>=l+m+mi0o&&npredp[nmatchp[nvp]]o>=l+m+mi0p))
          nBlossomContractp(nup,nvp);
        kelse kifp(npredp[nvp]o==-l+m+mi1p){
          npredp[nvp]o=nup;
          kifp(nmatchp[nvp]o>=l+m+mi0p) npushp(nmatchp[nvp]);
          kelsep{ nfinisho=nvp; kreturn n+nbtruep; p}
        p}
    p}
  p}
  kreturn nfoundp;
p}
k+ktvoid n+nfAugmentPathp(){
  k+ktint nuo=nfinishp,nvp,nwp;
  kwhilep(nuo>=l+m+mi0p){
    ↪ nvo=npredp[nup];nwo=nmatchp[nvp];nmatchp[nvp]o=nup;nmatchp[nup]o=nvp;nuo=nwp; p}
p}
k+ktvoid n+nfFindMaxMatchingp(){
  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip) nmatchp[nip]o=-l+m+mi1p;
  kforp(k+ktint nio=l+m+mi0p;nio<nnp;o++nip) kifp(nmatchp[nip]o==-l+m+mi1p)
    ↪ kifp(nFindAugmentingPathp(nip)) nAugmentPathp();
p}
```

Dominator Tree

```
nvectoro<k+ktinto> nprecp[nNp], nsuccp[nNp];
nvectoro<k+ktinto> nordp;
k+ktint nstampp, nvisp[nNp];
k+ktint nnump[nNp];
k+ktint nfap[nNp];
k+ktvoid n+nfdfsp(k+ktint nup) p{
```

```
  nvisp[nup] o= nstampp;
  nnump[nup] o= nordp.nsizep();
  nordp.npush_backp(nup);
  kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nsuccp[nup].nsizep(); o++nip) p{
    k+ktint nv o= nsuccp[nup][nip];
    kif p(nvisp[nvp] o!= nstampp) p{
      nfap[nvp] o= nup;
      ndfsp(nvp);
    p}
  p}
p}
k+ktint nfsp[nNp], nminsp[nNp], ndomp[nNp], nsemp[nNp];
k+ktint n+nffindp(k+ktint nup) p{
  kif p(nu o!= nfsp[nup]) p{
    k+ktint nv o= nfsp[nup];
    nfsp[nup] o= nfindp(nfsp[nup]);
    kif p(nminsp[nvp] o!= o-l+m+mi1 o&& nnump[nsemp[nminsp[nvp]]] o<
     ↪ nnump[nsemp[nminsp[nup]]]) p{
      nminsp[nup] o= nminsp[nvp];
    p}
  p}
  kreturn nfsp[nup];
p}
k+ktvoid n+nfmergep(k+ktint nup, k+ktint nvp) p{ nfsp[nup] o= nvp; p}
nvectoro<k+ktinto> nbufp[nNp];
k+ktint nbuf2p[nNp];
k+ktvoid n+nfmarkp(k+ktint nsourcep) p{
  nordp.nclearp();
  o++nstampp;
  ndfsp(nsourcep);
  kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nordp.nsizep(); o++nip) p{
    k+ktint nu o= nordp[nip];
    nfsp[nup] o= nup, nminsp[nup] o= o-l+m+mi1p, nbuf2p[nup] o= o-l+m+mi1p;
  p}
  kfor p(k+ktint ni o= p(k+ktintp)nordp.nsizep() o- l+m+mi1p; ni o> l+m+mi0p; o--nip) p{
    k+ktint nu o= nordp[nip], np o= nfap[nup];
    nsemp[nup] o= npp;
    kfor p(k+ktint nj o= l+m+mi0p; nj o< p(k+ktintp)nprecp[nup].nsizep(); o++njp) p{
      k+ktint nv o= nprecp[nup][njp];
      kif p(nusep[nvp] o!= nstampp) kcontinuep;
      kif p(nnump[nvp] o> nnump[nup]) p{
        nfindp(nvp); nv o= nsemp[nminsp[nvp]];
      p}
      kif p(nnump[nvp] o< nnump[nsemp[nup]]) p{
        nsemp[nup] o= nvp;
      p}
    p}
    nbufp[nsemp[nup]].npush_backp(nup);
    nminsp[nup] o= nup;
    nmergep(nup, npp);
    kwhile p(nbufp[npp].nsizep()) p{
      k+ktint nv o= nbufp[npp].nbackp();
      nbufp[npp].npop_backp();
      nfindp(nvp);
      kif p(nsemp[nvp] o== nsemp[nminsp[nvp]]) p{
        ndomp[nvp] o= nsemp[nvp];
      p} kelse p{
        nbuf2p[nvp] o= nminsp[nvp];
      p}
    p}
  p}
  ndomp[nordp[l+m+mi0p]] o= nordp[l+m+mi0p];
  kfor p(k+ktint ni o= l+m+mi0p; ni o< p(k+ktintp)nordp.nsizep(); o++nip) p{
    k+ktint nu o= nordp[nip];
    kif p(o~nbuf2p[nup]) p{
      ndomp[nup] o= ndomp[nbuf2p[nup]];
    p}
  p}
p}
```

无向图最小割

```
k+ktint ncostp[nmaxnp][nmaxnp],nseqp[nmaxnp],nlenp[nmaxnp],nnp,nmp,npopp,nansp;
k+ktbool nusedp[nmaxnp];
k+ktvoid n+nfInitp(){
  k+ktint nip,njp,nap,nbp,ncp;
  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;
  kforp(nio=l+m+mi0p;nio<nmp;nio++p){
    nscanfp(l+s"%d %d %d"p,o&nap,o&nbp,o&ncp); ncostp[nap][nbp]o+=ncp;
   ↪ ncostp[nbp][nap]o+=ncp;
  p}
  npopo=nnp; kforp(nio=l+m+mi0p;nio<nnp;nio++p) nseqp[nip]o=nip;
p}
k+ktvoid n+nfWorkp(){
  nanso=ninfp; k+ktint nip,njp,nkp,nlp,nmmp,nsump,npkp;
  kwhilep(npop o> l+m+mi1p){
    kforp(nio=l+m+mi1p;nio<npopp;nio++p) nusedp[nseqp[nip]]o=l+m+mi0p;
   ↪ nusedp[nseqp[l+m+mi0p]]o=l+m+mi1p;
    kforp(nio=l+m+mi1p;nio<npopp;nio++p)
   ↪ nlenp[nseqp[nip]]o=ncostp[nseqp[l+m+mi0p]][nseqp[nip]];
    npko=l+m+mi0p; nmmo=-ninfp; nko=-l+m+mi1p;
    kforp(nio=l+m+mi1p;nio<npopp;nio++p) kifp(nlenp[nseqp[nip]] o> nmmp){
   ↪ nmmo=nlenp[nseqp[nip]]; nko=nip; p}
    kforp(nio=l+m+mi1p;nio<npopp;nio++p){
      nusedp[nseqp[nlo=nkp]]o=l+m+mi1p;
      kifp(nio==npopo-l+m+mi2p) npko=nkp;
      kifp(nio==npopo-l+m+mi1p) kbreakp;
      nmmo=-ninfp;
      kforp(njo=l+m+mi1p;njo<npopp;njo++p) kifp(o!nusedp[nseqp[njp]])
        kifp((nlenp[nseqp[njp]]o+=ncostp[nseqp[nlp]][nseqp[njp]]) o> nmmp)
          nmmo=nlenp[nseqp[njp]], nko=njp;
    p}
    nsumo=l+m+mi0p;
    kforp(nio=l+m+mi0p;nio<npopp;nio++p) kifp(ni o!= nkp)
   ↪ nsumo+=ncostp[nseqp[nkp]][nseqp[nip]];
    nanso=nminp(nansp,nsump);
    kforp(nio=l+m+mi0p;nio<npopp;nio++p)
      ncostp[nseqp[nkp]][nseqp[nip]]o=ncostp[nseqp[nip]][nseqp[nkp]]o+=ncostp[nseqp[npkp]][nseqp[nip]
    nseqp[npkp]o=nseqp[o--npopp];
  p}
```

```
  nprintfp(l+s"%dl+s+se\nl+s"p,nansp);
p}
```

重口味费用流

```
k+ktint nSp, nTp, ntotFlowp, ntotCostp;

k+ktint ndisp[nNp], nslackp[nNp], nvisitp[nNp];

k+ktint n+nfmodlable p() p{
    k+ktint ndelta o= nINFp;
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p) p{
        kif p(o!nvisitp[nip] o&& nslackp[nip] o< ndeltap) ndelta o= nslackp[nip];
        nslackp[nip] o= nINFp;
    p}
    kif p(ndelta o== nINFp) kreturn l+m+mi1p;
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p)
        kif p(nvisitp[nip]) ndisp[nip] o+= ndeltap;
    kreturn l+m+mi0p;
p}

k+ktint n+nfdfs p(k+ktint nxp, k+ktint nflowp) p{
    kif p(nx o== nTp) p{
        ntotFlow o+= nflowp;
        ntotCost o+= nflow o* p(ndisp[nSp] o- ndisp[nTp]);
        kreturn nflowp;
    p}
    nvisitp[nxp] o= l+m+mi1p;
    k+ktint nleft o= nflowp;
    kfor p(k+ktint ni o= nep.nlastp[nxp]; o~nip; ni o= nep.nsuccp[nip])
        kif p(nep.ncapp[nip] o> l+m+mi0 o&& o!nvisitp[nep.notherp[nip]]) p{
            k+ktint ny o= nep.notherp[nip];
            kif p(ndisp[nyp] o+ nep.ncostp[nip] o== ndisp[nxp]) p{
                k+ktint ndelta o= ndfs p(nyp, nmin p(nleftp, nep.ncapp[nip]));
                nep.ncapp[nip] o-= ndeltap;
                nep.ncapp[ni o^ l+m+mi1p] o+= ndeltap;
                nleft o-= ndeltap;
                kif p(o!nleftp) p{ nvisitp[nxp] o= l+m+mi0p; kreturn nflowp; p}
            p} kelse p{
                nslackp[nyp] o= nmin p(nslackp[nyp], ndisp[nyp] o+ nep.ncostp[nip] o-
    ↪ ndisp[nxp]);
            p}
        p}
    kreturn nflow o- nleftp;
p}

npair o<k+ktintp, k+ktinto> nminCost p() p{
    ntotFlow o= l+m+mi0p; ntotCost o= l+m+mi0p;
    nfill p(ndis o+ l+m+mi1p, ndis o+ nT o+ l+m+mi1p, l+m+mi0p);
    kdo p{
        kdo p{
            nfill p(nvisit o+ l+m+mi1p, nvisit o+ nT o+ l+m+mi1p, l+m+mi0p);
        p} kwhile p(ndfs p(nSp, nINFp));
    p} kwhile p(o!nmodlable p());
    kreturn n+nfmake_pair p(ntotFlowp, ntotCostp);
p}
```

2-SAT

```
k+ktint nSp, nTp, ntotFlowp, ntotCostp;

k+ktint ndisp[nNp], nslackp[nNp], nvisitp[nNp];

k+ktint n+nfmodlable p() p{
    k+ktint ndelta o= nINFp;
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p) p{
        kif p(o!nvisitp[nip] o&& nslackp[nip] o< ndeltap) ndelta o= nslackp[nip];
        nslackp[nip] o= nINFp;
    p}
    kif p(ndelta o== nINFp) kreturn l+m+mi1p;
    kfor p(k+ktint ni o= l+m+mi1p; ni o<= nTp; nio++p)
        kif p(nvisitp[nip]) ndisp[nip] o+= ndeltap;
    kreturn l+m+mi0p;
p}

k+ktint n+nfdfs p(k+ktint nxp, k+ktint nflowp) p{
    kif p(nx o== nTp) p{
        ntotFlow o+= nflowp;
        ntotCost o+= nflow o* p(ndisp[nSp] o- ndisp[nTp]);
        kreturn nflowp;
    p}
    nvisitp[nxp] o= l+m+mi1p;
    k+ktint nleft o= nflowp;
    kfor p(k+ktint ni o= nep.nlastp[nxp]; o~nip; ni o= nep.nsuccp[nip])
        kif p(nep.ncapp[nip] o> l+m+mi0 o&& o!nvisitp[nep.notherp[nip]]) p{
            k+ktint ny o= nep.notherp[nip];
            kif p(ndisp[nyp] o+ nep.ncostp[nip] o== ndisp[nxp]) p{
                k+ktint ndelta o= ndfs p(nyp, nmin p(nleftp, nep.ncapp[nip]));
                nep.ncapp[nip] o-= ndeltap;
                nep.ncapp[ni o^ l+m+mi1p] o+= ndeltap;
                nleft o-= ndeltap;
                kif p(o!nleftp) p{ nvisitp[nxp] o= l+m+mi0p; kreturn nflowp; p}
            p} kelse p{
                nslackp[nyp] o= nmin p(nslackp[nyp], ndisp[nyp] o+ nep.ncostp[nip] o-
    ↪ ndisp[nxp]);
            p}
        p}
    kreturn nflow o- nleftp;
p}

npair o<k+ktintp, k+ktinto> nminCost p() p{
    ntotFlow o= l+m+mi0p; ntotCost o= l+m+mi0p;
    nfill p(ndis o+ l+m+mi1p, ndis o+ nT o+ l+m+mi1p, l+m+mi0p);
    kdo p{
        kdo p{
            nfill p(nvisit o+ l+m+mi1p, nvisit o+ nT o+ l+m+mi1p, l+m+mi0p);
        p} kwhile p(ndfs p(nSp, nINFp));
    p} kwhile p(o!nmodlable p());
    kreturn n+nfmake_pair p(ntotFlowp, ntotCostp);
p}
```

其他

Dancing Links

```
kstruct nNode p{
  nNode o*nlp, o*nrp, o*nup, o*ndp, o*ncolp;
  k+ktint nsizep, nline_nop;
  nNodep() p{
    nsize o= l+m+mi0p; nline_no o= o-l+m+mi1p;
    nl o= nr o= nu o= nd o= ncol o= n+nbNULLp;
  p}
p} o*nrootp;

k+ktvoid n+nfcoverp(nNode o*ncp) p{
  nco->nlo->nr o= nco->nrp; nco->nro->nl o= nco->nlp;
  kfor p(nNode o*nu o= nco->ndp; nu o!= ncp; nu o= nuo->ndp)
    kfor p(nNode o*nv o= nuo->nrp; nv o!= nup; nv o= nvo->nrp) p{
      nvo->ndo->nu o= nvo->nup;
      nvo->nuo->nd o= nvo->ndp;
      o-- nvo->ncolo->nsizep;
    p}
p}

k+ktvoid n+nfuncoverp(nNode o*ncp) p{
  kfor p(nNode o*nu o= nco->nup; nu o!= ncp; nu o= nuo->nup) p{
    kfor p(nNode o*nv o= nuo->nlp; nv o!= nup; nv o= nvo->nlp) p{
      o++ nvo->ncolo->nsizep;
      nvo->nuo->nd o= nvp;
      nvo->ndo->nu o= nvp;
    p}
  p}
  nco->nlo->nr o= ncp; nco->nro->nl o= ncp;
p}

nstdo::nvectoro<k+ktinto> nanswerp;
k+ktbool n+nfsearchp(k+ktint nkp) p{
  kif p(nrooto->nr o== nrootp) kreturn n+nbtruep;
  nNode o*nr o= n+nbNULLp;
  kfor p(nNode o*nu o= nrooto->nrp; nu o!= nrootp; nu o= nuo->nrp)
    kif p(nr o== n+nbNULL o|| nuo->nsize o< nro->nsizep)
      nr o= nup;
  kif p(nr o== n+nbNULL o|| nro->nsize o== l+m+mi0p) kreturn n+nbfalsep;
  kelse p{
    ncoverp(nrp);
    k+ktbool nsucc o= n+nbfalsep;
    kfor p(nNode o*nu o= nro->ndp; nu o!= nr o&& o!nsuccp; nu o= nuo->ndp) p{
      nanswerp.npush_backp(nuo->nline_nop);
      kfor p(nNode o*nv o= nuo->nrp; nv o!= nup; nv o= nvo->nrp)  c+c1// Cover row
        ncoverp(nvo->ncolp);
      nsucc o|= nsearchp(nk o+ l+m+mi1p);
      kfor p(nNode o*nv o= nuo->nlp; nv o!= nup; nv o= nvo->nlp)
        nuncoverp(nvo->ncolp);
      kif p(o!nsuccp) nanswerp.npop_backp();
    p}
    nuncoverp(nrp);
    kreturn nsuccp;
  p}
p}

k+ktbool nentryp[nCRp][nCCp];
nNode o*nwhop[nCRp][nCCp];
k+ktint ncrp, nccp;

k+ktvoid n+nfconstructp() p{
  nroot o= knew nNodep();
  nNode o*nlast o= nrootp;
  kfor p(k+ktint ni o= l+m+mi0p; ni o< nccp; o++ nip) p{
    nNode o*nu o= knew nNodep();
    nlasto->nr o= nup; nuo->nl o= nlastp;
    nNode o*nv o= nup; nuo->nline_no o= nip;
    nlast o= nup;
    kfor p(k+ktint nj o= l+m+mi0p; nj o< ncrp; o++ njp)
      kif p(nentryp[njp][nip]) p{
        o++ nuo->nsizep;
        nNode o*ncur o= knew nNodep();
        nwhop[njp][nip] o= ncurp;
        ncuro->nline_no o= njp;
        ncuro->ncol o= nup;
        ncuro->nu o= nvp; nvo->nd o= ncurp;
        nv o= ncurp;
      p}
    nvo->nd o= nup; nuo->nu o= nvp;
  p}
  nlasto->nr o= nrootp; nrooto->nl o= nlastp;
  kfor p(k+ktint nj o= l+m+mi0p; nj o< ncrp; o++ njp) p{
    nNode o*nlast o= n+nbNULLp;
    kfor p(k+ktint ni o= ncc o- l+m+mi1p; ni o>= l+m+mi0p; o-- nip)
      kif p(nentryp[njp][nip]) p{
        nlast o= nwhop[njp][nip];
        kbreakp;
      p}
    kfor p(k+ktint ni o= l+m+mi0p; ni o< nccp; o++ nip)
      kif p(nentryp[njp][nip]) p{
        nlasto->nr o= nwhop[njp][nip];
        nwhop[njp][nip]o->nl o= nlastp;
        nlast o= nwhop[njp][nip];
      p}
  p}
p}

k+ktvoid n+nfdestructp() p{
  kfor p(nNode o*nu o= nrooto->nrp; nu o!= nrootp; p) p{
    kfor p(nNode o*nv o= nuo->ndp; nv o!= nup; p) p{
      nNode o*nnxt o= nvo->ndp;
      kdeletep(nvp);
      nv o= nnxtp;
    p}
    nNode o*nnxt o= nuo->nrp;
    kdeletep(nup); nu o= nnxtp;
  p}
```

```
  kdelete nrootp;
p}
```

蔡勒公式

0 for Sunday. Day and month is 1-based.

```
k+ktint n+nfzellerp(k+ktint nyp,k+ktint nmp,k+ktint ndp) p{
  kif p(nmo<=l+m+mi2p) nyo--p,nmo+=l+m+mi12p; k+ktint nco=nyo/l+m+mi100p; nyo%=l+m+mi100p;
  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;
  kif p(nwo<l+m+mi0p) nwo+=l+m+mi7p; kreturnp(nwp);
p}
```

技巧

真正的释放 STL 容器内存空间

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

无敌的大整数相乘取模

Time complexity $O(1)$.

```
c+c1// 需要保证 x 和 y 非负
k+ktlong k+ktlong n+nfmultp(k+ktlong k+ktlong nxp, k+ktlong k+ktlong nyp, k+ktlong k+ktlong
   ↪ nMODNp) p{
  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;
  kreturn nt o< l+m+mi0 o? nt o+ n+nlMODN p: ntp;
p}
```

无敌的读入优化

```
c+c1// getchar() 读入优化 << 关同步 cin << 此优化
c+c1// 用 isdigit() 会小幅变慢
c+c1// 返回 false 表示读到文件尾
knamespace nReader p{
    kconst k+ktint nL o= p(l+m+mi1 o<< l+m+mi15p) o+ l+m+mi5p;
    k+ktchar nbufferp[nLp], o*nSp, o*nTp;
    k+kr__inline k+ktbool n+nfgetcharp(k+ktchar o&nchp) p{
        kif p(nS o== nTp) p{
            nT o= p(nS o= nbufferp) o+ nfreadp(nbufferp, l+m+mi1p, nLp, nstdinp);
            kif p(nS o== nTp) p{
        nch o= nEOFp;
        kreturn n+nbfalsep;
      p}
        p}
    nch o= o*nSo++p;
    kreturn n+nbtruep;
    p}
    k+kr__inline k+ktbool n+nfgetintp(k+ktint o&nxp) p{
    k+ktchar nchp; k+ktbool nneg o= l+m+mi0p;
    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;
    kif p(nch o== nEOFp) kreturn n+nbfalsep;
    nx o= nch o- l+s+sc'0'p;
    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;
    kif p(nnegp) nx o= o-nxp;
    kreturn n+nbtruep;
    p}
p}
```

梅森旋转算法

High quality pseudorandom number generator, twice as efficient as rand() with -O2. C++11 required.

```
c+cp#include c+cpf<random>

k+ktint n+nfmainp() p{
    nstdo::nmt19937 ngp(nseedp);  c+c1// std::mt19937_64
    nstdo::ncout o<< ngp() o<< nstdo::nendlp;
p}
```

提示

控制 cout 输出实数精度

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

vimrc

```
set nu
set sw=4
set sts=4
set ts=4
syntax on
set cindent
```

让 make 支持 c ++ 11

In .bashrc or whatever:

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

tuple 相关

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

线性规划转对偶

$$\begin{array}{l}\text{maximize } \mathbf{c}^T\mathbf{x} \\ \text{subject to } \mathbf{Ax} \le \mathbf{b}, \mathbf{x} \ge 0\end{array} \iff \begin{array}{l}\text{minimize } \mathbf{y}^T\mathbf{b} \\ \text{subject to } \mathbf{y}^T\mathbf{A} \ge \mathbf{c}^T, \mathbf{y} \ge 0\end{array}$$

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

```
k+knimport n+nnjava.io.*o;
k+knimport n+nnjava.util.*o;
k+knimport n+nnjava.math.*o;

k+kdpublic k+kdclass n+ncMain o{
  k+kdstatic k+ktint n+nfgeto(k+ktchar nco) o{
    kif o(nc o<= l+s+sc'9'o)
      kreturn nc o- l+s+sc'0'o;
    kelse kif o(nc o<= l+s+sc'Z'o)
      kreturn nc o- l+s+sc'A' o+ l+m+mi10o;
    kelse
      kreturn nc o- l+s+sc'a' o+ l+m+mi36o;
  o}
  k+kdstatic k+ktchar n+nfgeto(k+ktint nxo) o{
    kif o(nx o<= l+m+mi9o)
      kreturn o(k+ktcharo)(nx o+ l+s+sc'0'o);
    kelse kif o(nx o<= l+m+mi35o)
      kreturn o(k+ktcharo)(nx o- l+m+mi10 o+ l+s+sc'A'o);
    kelse
      kreturn o(k+ktcharo)(nx o- l+m+mi36 o+ l+s+sc'a'o);
  o}
  k+kdstatic nBigInteger n+nfgeto(nString nso, nBigInteger nxo) o{
    nBigInteger nans o= nBigIntegero.n+navalueOfo(l+m+mi0o), nnow o=
   ↪ nBigIntegero.n+navalueOfo(l+m+mi1o);
    kfor o(k+ktint ni o= nso.n+nalengtho() o- l+m+mi1o; ni o>= l+m+mi0o; nio--) o{
      nans o=
   ↪ nanso.n+naaddo(nnowo.n+namultiplyo(nBigIntegero.n+navalueOfo(ngeto(nso.n+nacharAto(nio)))));
      nnow o= nnowo.n+namultiplyo(nxo);
    o}
    kreturn nanso;
  o}
  k+kdpublic k+kdstatic k+ktvoid n+nfmaino(nString o[] nargso) o{
    nScanner ncin o= knew nScannero(knew nBufferedInputStreamo(nSystemo.n+naino));
    kfor o(; o; o) o{
      nBigInteger nx o= ncino.n+nanextBigIntegero();
      kif o(nxo.n+nacompareToo(nBigIntegero.n+navalueOfo(l+m+mi0o)) o== l+m+mi0o)
        kbreako;
      nString ns o= ncino.n+nanexto(), nt o= ncino.n+nanexto(), nr o= l+s""o;
      nBigInteger nans o= ngeto(nso, nxo).n+namodo(ngeto(nto, nxo));
      kif o(nanso.n+nacompareToo(nBigIntegero.n+navalueOfo(l+m+mi0o)) o== l+m+mi0o)
        nr o= l+s"0"o;
      kfor o(; nanso.n+nacompareToo(nBigIntegero.n+navalueOfo(l+m+mi0o)) o> l+m+mi0o;) o{
        nr o= ngeto(nanso.n+namodo(nxo).n+naintValueo()) o+ nro;
        nans o= nanso.n+nadivideo(nxo);
      o}
      nSystemo.n+naouto.n+naprintlno(nro);
    o}
  o}
o}

c+c1// Arrays
k+ktint nao[];
o.n+nafillo(nao[, k+ktint nfromIndexo, k+ktint ntoIndexo],nvalo); o| o.n+nasorto(nao[,
   ↪ k+ktint nfromIndexo, k+ktint ntoIndexo])
c+c1// String
nString nso;
o.n+nacharAto(k+ktint nio); o| ncompareToo(nStringo) o| ncompareToIgnoreCase o() o|
   ↪ ncontainso(nStringo) o|
nlength o() o| nsubstringo(k+ktint nlo, k+ktint nleno)
c+c1// BigInteger
o.n+naabso() o| o.n+naaddo() o| nbitLength o() o| nsubtract o() o| ndivide o() o| nremainder
   ↪ o() o| ndivideAndRemainder o() o| nmodPowo(nbo, nco) o|
npowo(k+ktinto) o| nmultiply o() o| ncompareTo o() o|
ngcdo() o| nintValue o() o| nlongValue o() o| nisProbablePrimeo(k+ktint nco) o(l+m+mi1 o-
   ↪ l+m+mi1o/l+m+mi2o^nco) o|
nnextProbablePrime o() o| nshiftLefto(k+ktinto) o| nvalueOf o()
c+c1// BigDecimal
o.n+naROUND_CEILING o| nROUND_DOWN_FLOOR o| nROUND_HALF_DOWN o| nROUND_HALF_EVEN o|
   ↪ nROUND_HALF_UP o| nROUND_UP
o.n+nadivideo(nBigDecimal nbo, k+ktint nscale o, k+ktint nround_modeo) o| ndoubleValue o() o|
   ↪ nmovePointLefto(k+ktinto) o| npowo(k+ktinto) o|
nsetScaleo(k+ktint nscale o, k+ktint nround_modeo) o| nstripTrailingZeros o()
c+c1// StringBuilder
nStringBuilder nsb o= knew nStringBuilder o();
nsbo.n+naappendo(nelemo) o| nouto.n+naprintlno(nsbo)
```