

1 Computational_Geometry

1.1 formula.tex

Pick 公式給定頂點坐標均是整點的簡單多邊形，有：面積 = 內部格點數 + 邊上格點數 / 2 - 1

1.2 Geometry.cpp

```
1 template<typename T>
2 struct point{
3     T x,y;
4     point(){ }
5     point(const T&x,const T&y):x(x),y(y){ }
6     point operator+(const point &b)const{
7         return point(x+b.x,y+b.y);
8     }
9     point operator-(const point &b)const{
10        return point(x-b.x,y-b.y);
11    }
12    point operator*(const T &b)const{return
13        point(x*b,y*b);
14    }
15    point operator/(const T &b)const{return
16        point(x/b,y/b);
17    }
18    bool operator==(const point &b)const{
19        return x==b.x&&y==b.y;
20    }
21    T dot(const point &b)const{
22        return x*b.x+y*b.y;
23    }
24    T cross(const point &b)const{
25        return x*b.y-y*b.x;
26    }
27    point normal()const{/*求法向量*/
28        return point(-y,x);
29    }
30    T abs2()const{/*向量長度的平方*/
31        return dot(*this);
32    }
33    T rad(const point &b)const{/*兩向量的弧度(
34        夾角)*/
35        return fabs(atan2(fabs(cross(b)),dot(b)));
36    }
37 };
38 template<typename T>
39 struct line{
40     line(){ }
41     point<T> p1,p2;
42     T a,b,c; /*ax+by+c=0*/
43     line(const point<T>&x,const point<T>&y):p1(
44         x),p2(y){ }
45     void pton(){/*轉成一般式*/
46         a=p1.y-p2.y;
47         b=p2.x-p1.x;
48         c=-a*p1.x-b*p1.y;
49     }
50     T cross(const point<T> &p)const{/*點和有向
51         直線的關係，>0左邊，=0在線上<0右邊*/
52         return (p2-p1).cross(p-p1);
53     }
```

```
54     bool point_on_segment(const point<T>&p)
55     const{/*點是否線段上*/
56         return cross(p)==0&&(p1-p1).dot(p2-p1)<=0;
57     }
58     T dis2(const point<T> &p,bool is_segment
59         =0)const{/*點跟直線/線段的距離平方*/
60         point<T> v=p2-p1,v1=p-p1;
61         if(is_segment){
62             point<T> v2=p-p2;
63             if(v.dot(v1)<=0)return v1.abs2();
64             if(v.dot(v2)>=0)return v2.abs2();
65         }
66         T tmp=v.cross(v1);
67         return tmp*tmp/v.abs2();
68     }
69     point<T> projection(const point<T> &p)
70     const{/*點對直線的投影
71         點對直線的鏡射*/
72         point<T> n=(p2-p1).normal();
73         return p-n*(p-p1).dot(n)/n.abs2();
74     }
75     point<T> mirror(const point<T> &p)const{/*
76         點對直線的鏡射*/
77         /*要先呼叫 pton 轉成一般式*/
78         point<T> ans;
79         T d=a*a+b*b;
80         ans.x=(b*b*p.x-a*a*p.x-2*a*b*p.y-2*a*c)/
81             d;
82         ans.y=(a*a*p.y-b*b*p.y-2*a*b*p.x-2*b*c)/
83             d;
84         return ans;
85     }
86     bool equal(const line &l)const{/*直線相等
87         */
88         return cross(l.p1)==0&&cross(l.p2)==0;
89     }
90     bool parallel(const line &l)const{/*直線平
91         行*/
92         return (p1-p2).cross(l.p1-l.p2)==0;
93     }
94     bool cross_seg(const line &l)const{/*直線
95         是否交線段*/
96         return (p2-p1).cross(l.p1)*(p2-p1).cross
97             (l.p2)<=0;
98     }
99     char line_intersect(const line &l)const{/*
100        直線相交情況，-1無限多點，1交於一點，0
101        不相交*/
102        return parallel(l)?(cross(l.p1)==0?-1:0):
103            1;
104    }
105    char seg_intersect(const line &l)const{/*
106        線段相交情況，-1無限多點，1交於一點，0
107        不相交*/
108        T c1=(p2-p1).cross(l.p1-p1);
109        T c2=(p2-p1).cross(l.p2-p1);
110        T c3=(l.p2-l.p1).cross(p1-l.p1);
111        T c4=(l.p2-l.p1).cross(p2-l.p1);
112        if(c1==0&&c2==0){
113            if(p1==l.p1&&(p2-p1).dot(l.p2)<=0)
114                return 1;
115            if(p1==l.p2&&(p2-p1).dot(l.p1)<=0)
116                return 1;
117            return 0;
118        }
119        if(c1*c2<=0&&c3*c4<=0)return 1;
120        return 0;
121    }
122    point<T> line_intersection(const line &l)
123    const{/*直線交點*/
124        point<T> a=p2-p1,b=l.p2-l.p1,s=l.p1-p1;
125        //if(a.cross(b)==0)return INF;
126        return p1+a*s.cross(b)/a.cross(b);
127    }
128    point<T> seg_intersection(const line &l)
129    const{/*線段交點*/
130        T c1=(p2-p1).cross(l.p1-p1);
131        T c2=(p2-p1).cross(l.p2-p1);
132        T c3=(l.p2-l.p1).cross(p1-l.p1);
133        T c4=(l.p2-l.p1).cross(p2-l.p1);
134        if(c1==0&&c2==0){
135            if(p1==l.p1&&(p2-p1).dot(l.p2)<=0)
136                return p1;
137            if(p1==l.p2&&(p2-p1).dot(l.p1)<=0)
138                return p1;
139            if(p2==l.p1&&(p1-p2).dot(l.p2)<=0)
140                return p2;
141            if(p2==l.p2&&(p1-p2).dot(l.p1)<=0)
142                return p2;
143        }
144        else if(c1*c2<=0&&c3*c4<=0)return
145            line_intersection(l);
146        //return INF;
147    }
148 };
149 template<typename T>
150 struct polygon{
151     polygon(){ }
152     vector<point<T> > p; //逆時針順序
153     T area()const{/*多邊形面積*/
154         T ans=0;
155         for(int i=p.size()-1,j=0;j<(int)p.size()
156             ;i=j++){
157             ans+=p[i].cross(p[j]);
158         }
159         return ans/2;
160     }
161     point<T> center_of_mass()const{/*多邊形重
162         心*/
163         T cx=0,cy=0,w=0;
164         for(int i=p.size()-1,j=0;j<(int)p.size()
165             ;i=j++){
166             T a=p[i].cross(p[j]);
167             cx+=(p[i].x+p[j].x)*a;
168             cy+=(p[i].y+p[j].y)*a;
169             w+=a;
170         }
171         return point<T>(cx/3/w,cy/3/w);
172     }
173     char ahas(const point<T>& t)const{/*點是否
174         在簡單多邊形內，是的話回傳1、在邊上回
175         傳-1、否則回傳0*/
176         bool c=0;
177         for(int i=0,j=p.size()-1;i<p.size();j=i
178             ++)
```

```
179         if(line<T>(p[i],p[j]).point_on_segment
180             (t))return -1;
181         else if((p[i].y>t.y)!=p[j].y>t.y)&&
182             t.x<(p[j].x-p[i].x)*(t.y-p[i].y)/(p[j]
183                 .y-p[i].y)+p[i].x)
184             c=1;
185         return c;
186     }
187     char point_in_convex(const point<T>&x)
188     const{
189         int l=1,r=(int)p.size()-2;
190         while(l<r){/*點是否在凸多邊形內，是的話
191             回傳1、在邊上回傳-1、否則回傳0*/
192             int mid=(l+r)/2;
193             T a1=(p[mid]-p[0]).cross(x-p[0]);
194             T a2=(p[mid+1]-p[0]).cross(x-p[0]);
195             if(a1>=0&&a2<=0){
196                 T res=(p[mid+1]-p[mid]).cross(x-p[
197                     mid]);
198                 return res>0?-1:(res>0?-1:0);
199             }
200             else if(a1<0)r=mid-1;
201             else l=mid+1;
202         }
203         return 0;
204     }
205     polygon cut(const line<T> &l)const{/*凸包
206         對直線切割，得到直線l左側的凸包*/
207     polygon ans;
208     for(int n=p.size(),i=n-1,j=0;j<n;i=j++){
209         if(l.cross(p[i])>=0){
210             ans.p.push_back(p[i]);
211             if(l.cross(p[j])<0)
212                 ans.p.push_back(l.
213                     line_intersection(line<T>(p[i]
214                         ,p[j])));
215         }
216         else if(l.cross(p[j])>0)
217             ans.p.push_back(l.line_intersection(
218                 line<T>(p[i],p[j])));
219     }
220     return ans;
221 }
222 static bool graham_cmp(const point<T>& a,
223     const point<T>& b){
224     return (a.x<b.x)||a.x==b.x&&a.y<b.y; /*
225         凸包排序函數*/
226 }
227 void graham(vector<point<T> > &s){/*凸包*/
228     sort(s.begin(),s.end(),graham_cmp);
229     p.resize(s.size()+1);
230     int m=0;
231     for(int i=0;i<(int)s.size();i++){
232         while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]
233             -p[m-2])<=0)--m;
234         p[m++]=s[i];
235     }
236     for(int i=s.size()-2,t=m+1;i>=0;--i){
237         while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]
238             -p[m-2])<=0)--m;
239         p[m++]=s[i];
240     }
241     if(s.size())>1--m;
242     p.resize(m);
243 }
244 inline static char sign(const point<T>&t){
245     return (t.y==0?t.x:t.y)<0;
```

```

188 }
189 inline static bool angle_cmp(const line<T
190 >& A, const line<T>& B){
191     point<T> a=A.p2-A.p1, b=B.p2-B.p1;
192     return sign(a)<sign(b)|| (sign(a)==sign(b)
193         &&a.cross(b)>0);
194 }
195 int halfplane_intersection(vector<line<T>
196 > &s){//半平面交
197     sort(s.begin(), s.end(), angle_cmp); //線段
198     //左側為該線段半平面
199     int L,R,n=s.size();
200     vector<point<T> > px(n);
201     vector<line<T> > q(n);
202     q[L=R=0]=s[0];
203     for(int i=1; i<n; ++i){
204         while(L<R&&s[i].cross(px[R-1])<=0)--R;
205         while(L<R&&s[i].cross(px[L])<=0)++L;
206         q[++R]=s[i];
207         if(q[R].parallel(q[R-1])){
208             --R;
209             if(q[R].cross(s[i].p1)>0)q[R]=s[i];
210         }
211         if(L<R)px[R-1]=q[R-1].
212             line_intersection(q[R]);
213         while(L<R&&q[L].cross(px[R-1])<=0)--R;
214         p.clear();
215         if(R-L<=1)return 0;
216         px[R]=q[R].line_intersection(q[L]);
217         for(int i=L; i<=R; ++i)p.push_back(px[i]);
218         return R-L+1;
219     }
220 };
221 template<typename T>
222 struct triangle{
223     point<T> a,b,c;
224     triangle(){
225         triangle(const point<T> &a, const point<T>
226             &b, const point<T> &c):a(a),b(b),c(c){
227             T area(){const{
228                 T t=(b-a).cross(c-a)/2;
229                 return t>0?t:-t;
230             }
231             point<T> barycenter(){const{ //重心 *
232                 return (a+b+c)/3;
233             }
234             point<T> circumcenter(){const{ //外心 *
235                 static line<T> u,v;
236                 u.p1=(a+b)/2;
237                 u.p2=point<T>(u.p1.x-a.y+b.y, u.p1.y+a.x-
238                     b.x);
239                 v.p1=(a+c)/2;
240                 v.p2=point<T>(v.p1.x-a.y+c.y, v.p1.y+a.x-
241                     c.x);
242                 return u.line_intersection(v);
243             }
244             point<T> incenter(){const{ //內心 · 用到根號
245                 *
246                 T A=sqrt((b-c).abs2()), B=sqrt((a-c).abs2
247                     ()), C=sqrt((a-b).abs2());
248                 return point<T>(A*a.x+B*b.x+C*c.x, A*a.y+
249                     B*b.y+C*c.y)/(A+B+C);
250             }
251             point<T> perpercenter(){const{ //垂心 *
252                 return barycenter()*3-circumcenter()*2;
253             }
254             template<typename T>
255             struct point3D{
256                 T x,y,z;
257                 point3D(){
258                     point3D(const T&x, const T&y, const T&z):x(x
259                         ),y(y),z(z){
260                     point3D operator+(const point3D &b) const{
261                         return point3D(x+b.x, y+b.y, z+b.z);
262                     }
263                     point3D operator-(const point3D &b) const{
264                         return point3D(x-b.x, y-b.y, z-b.z);
265                     }
266                     point3D operator*(const T &b) const{
267                         return point3D(x*b, y*b, z*b);
268                     }
269                     point3D operator/(const T &b) const{
270                         return point3D(x/b, y/b, z/b);
271                     }
272                     bool operator==(const point3D &b) const{
273                         return x==b.x&&y==b.y&&z==b.z;
274                     }
275                     T dot(const point3D &b) const{
276                         return x*b.x+y*b.y+z*b.z;
277                     }
278                     point3D cross(const point3D &b) const{
279                         return point3D(y*b.z-z*b.y, z*b.x-x*b.z, x
280                             *b.y-y*b.x);
281                     }
282                     T abs2(){const{ //向量長度的平方 *
283                         return dot(*this);
284                     }
285                     T area2(const point3D &b) const{ //和b、原點
286                         //圍成面積的平方
287                         return cross(b).abs2()/4;
288                     }
289             };
290             template<typename T>
291             struct line3D{
292                 point3D<T> p1,p2;
293                 line3D(){
294                     line3D(const point3D<T> &p1, const point3D<
295                         T> &p2):p1(p1),p2(p2){
296                     T dis2(const point3D<T> &p, bool is_segment
297                         =0) const{ //點跟直線/線段的距離平方 *
298                         point3D<T> v=p2-p1, v1=p-p1;
299                         if(is_segment){
300                             point3D<T> v2=p-p2;
301                             if(v.dot(v1)<=0) return v1.abs2();
302                             if(v.dot(v2)>=0) return v2.abs2();
303                         }
304                         point3D<T> tmp=v.cross(v1);
305                         return tmp.abs2()/(v.abs2());
306                     }
307                     pair<point3D<T>, point3D<T> > closest_pair(
308                         const line3D<T> &l) const{
309                         point3D<T> v1=(p1-p2), v2=(l.p1-l.p2);
310                         point3D<T> N=v1.cross(v2), ab=(p1-l.p1);
311                         //if(N.abs2()==0) return NULL; 平行或重合
312                         T tmp=N.dot(ab), ans=tmp*tmp/N.abs2(); //
313                         //最近點對距離
314                     }
315                     point3D<T> d1=p2-p1, d2=l.p2-l.p1, D=d1.
316                         cross(d2);
317                     T t1=((l.p1-p1).cross(d2)).dot(D)/D.abs2
318                         ();
319                     T t2=((l.p1-p1).cross(d1)).dot(D)/D.
320                         abs2();
321                     return make_pair(p1+d1*t1, l.p1+d2*t2);
322                 }
323                 bool same_side(const point3D<T> &a, const
324                     point3D<T> &b) const{
325                     return (p2-p1).dot((p2-p1).
326                         cross(b-p1))>0;
327                 }
328             };
329             template<typename T>
330             struct plane{
331                 point3D<T> p0,n; //平面上的點和法向量
332                 plane(){
333                     plane(const point3D<T> &p0, const point3D<T>
334                         > &n):p0(p0),n(n){
335                     T dis2(const point3D<T> &p) const{ //點到平
336                         //面距離的平方
337                     T tmp=(p-p0).dot(n);
338                     return tmp*tmp/n.abs2();
339                     }
340                     point3D<T> projection(const point3D<T> &p)
341                         const{
342                     return p-n*(p-p0).dot(n)/n.abs2();
343                     }
344                     point3D<T> line_intersection(const line3D<
345                         T> &l) const{
346                     T tmp=n.dot(l.p2-l.p1); //等於0表示平行或
347                         //重合該平面
348                     return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/
349                         tmp);
350                     }
351                     line3D<T> plane_intersection(const plane &
352                         p1) const{
353                     point3D<T> e=n.cross(p1.n), v=n.cross(e);
354                     T tmp=p1.n.dot(v); //等於0表示平行或重合
355                         //該平面
356                     point3D<T> q=p0+(v*(p1.n.dot(p1.p0-p0))/
357                         tmp);
358                     return line3D<T>(q, q+e);
359                     }
360             };
361             template<typename T>
362             struct triangle3D{
363                 point3D<T> a,b,c;
364                 triangle3D(){
365                     triangle3D(const point3D<T> &a, const
366                         point3D<T> &b, const point3D<T> &c):a(a
367                         ),b(b),c(c){
368                     bool point_in(const point3D<T> &p) const{ //
369                         //點在該平面上的投影在三角形中
370                     return line3D<T>(b,c).same_side(p,a)&&
371                         line3D<T>(a,c).same_side(p,b)&&
372                         line3D<T>(a,b).same_side(p,c);
373                     }
374                 }
375             };
376             template<typename T>
377             struct tetrahedron{ //四面體
378                 point3D<T> a,b,c,d;
379                 tetrahedron(){
380                     tetrahedron(const point3D<T> &a, const
381                         point3D<T> &b, const point3D<T> &c,
382                         const point3D<T> &d):a(a),b(b),c(c),d(
383                             d){
384                     T volume6(){const{ //體積的六倍
385                         return (d-a).dot((b-a).cross(c-a));
386                     }
387                     point3D<T> centroid(){const{
388                         return (a+b+c+d)/4;
389                     }
390                     bool point_in(const point3D<T> &p) const{
391                         return triangle3D<T>(a,b,c).point_in(p)
392                             &&triangle3D<T>(c,d,a).point_in(p);
393                     }
394                 };
395                 template<typename T>
396                 struct convexhull3D{
397                     static const int MAXN=105;
398                     struct face{
399                         int a,b,c;
400                         bool use;
401                         face(){
402                             face(int a, int b, int c):a(a),b(b),c(c),
403                                 use(1){
404                             };
405                         vector<point3D<T> > pt;
406                         vector<face> fc;
407                         int fid[MAXN][MAXN];
408                         static bool point_cmp(const point3D<T> &a,
409                             const point3D<T> &b){
410                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
411                                 y==b.y&&a.z<b.z)));
412                         }
413                         bool outside(int p, int a, int b, int c) const
414                             {
415                         return tetrahedron<T>(pt[a], pt[b], pt[c],
416                             pt[p]).volume6()<0;
417                         }
418                         bool outside(int p, int f) const{ return
419                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
420                         void add_face(int a, int b, int c, int p){
421                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
422                                 fid[a][c]=fc.size(), fc.push_back(
423                                     face(c, b, a));
424                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
425                                 size(), fc.push_back(face(a, b, c));
426                         }
427                         bool dfs(int p, int f){
428                             if(!fc[f].use) return true;
429                             if(outside(p, f)){
430                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
431                                 fc[f].use=false;
432                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
433                                     ;
434                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
435                                     ;
436                                 return true;
437                             } else return false;
438                         }
439                         void build(){
440                             bool ok=false;
441                             fc.clear();
442                             sort(pt.begin(), pt.end(), point_cmp);
443                         }
444                     };
445                 };
446             };
447             template<typename T>
448             struct tetrahedron{ //四面體
449                 point3D<T> a,b,c,d;
450                 tetrahedron(){
451                     tetrahedron(const point3D<T> &a, const
452                         point3D<T> &b, const point3D<T> &c,
453                         const point3D<T> &d):a(a),b(b),c(c),d(
454                             d){
455                     T volume6(){const{ //體積的六倍
456                         return (d-a).dot((b-a).cross(c-a));
457                     }
458                     point3D<T> centroid(){const{
459                         return (a+b+c+d)/4;
460                     }
461                     bool point_in(const point3D<T> &p) const{
462                         return triangle3D<T>(a,b,c).point_in(p)
463                             &&triangle3D<T>(c,d,a).point_in(p);
464                     }
465                 };
466                 template<typename T>
467                 struct convexhull3D{
468                     static const int MAXN=105;
469                     struct face{
470                         int a,b,c;
471                         bool use;
472                         face(){
473                             face(int a, int b, int c):a(a),b(b),c(c),
474                                 use(1){
475                             };
476                         vector<point3D<T> > pt;
477                         vector<face> fc;
478                         int fid[MAXN][MAXN];
479                         static bool point_cmp(const point3D<T> &a,
480                             const point3D<T> &b){
481                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
482                                 y==b.y&&a.z<b.z)));
483                         }
484                         bool outside(int p, int a, int b, int c) const
485                             {
486                         return tetrahedron<T>(pt[a], pt[b], pt[c],
487                             pt[p]).volume6()<0;
488                         }
489                         bool outside(int p, int f) const{ return
490                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
491                         void add_face(int a, int b, int c, int p){
492                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
493                                 fid[a][c]=fc.size(), fc.push_back(
494                                     face(c, b, a));
495                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
496                                 size(), fc.push_back(face(a, b, c));
497                         }
498                         bool dfs(int p, int f){
499                             if(!fc[f].use) return true;
500                             if(outside(p, f)){
501                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
502                                 fc[f].use=false;
503                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
504                                     ;
505                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
506                                     ;
507                                 return true;
508                             } else return false;
509                         }
510                         void build(){
511                             bool ok=false;
512                             fc.clear();
513                             sort(pt.begin(), pt.end(), point_cmp);
514                         }
515                     };
516                 };
517             };
518             template<typename T>
519             struct tetrahedron{ //四面體
520                 point3D<T> a,b,c,d;
521                 tetrahedron(){
522                     tetrahedron(const point3D<T> &a, const
523                         point3D<T> &b, const point3D<T> &c,
524                         const point3D<T> &d):a(a),b(b),c(c),d(
525                             d){
526                     T volume6(){const{ //體積的六倍
527                         return (d-a).dot((b-a).cross(c-a));
528                     }
529                     point3D<T> centroid(){const{
530                         return (a+b+c+d)/4;
531                     }
532                     bool point_in(const point3D<T> &p) const{
533                         return triangle3D<T>(a,b,c).point_in(p)
534                             &&triangle3D<T>(c,d,a).point_in(p);
535                     }
536                 };
537                 template<typename T>
538                 struct convexhull3D{
539                     static const int MAXN=105;
540                     struct face{
541                         int a,b,c;
542                         bool use;
543                         face(){
544                             face(int a, int b, int c):a(a),b(b),c(c),
545                                 use(1){
546                             };
547                         vector<point3D<T> > pt;
548                         vector<face> fc;
549                         int fid[MAXN][MAXN];
550                         static bool point_cmp(const point3D<T> &a,
551                             const point3D<T> &b){
552                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
553                                 y==b.y&&a.z<b.z)));
554                         }
555                         bool outside(int p, int a, int b, int c) const
556                             {
557                         return tetrahedron<T>(pt[a], pt[b], pt[c],
558                             pt[p]).volume6()<0;
559                         }
560                         bool outside(int p, int f) const{ return
561                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
562                         void add_face(int a, int b, int c, int p){
563                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
564                                 fid[a][c]=fc.size(), fc.push_back(
565                                     face(c, b, a));
566                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
567                                 size(), fc.push_back(face(a, b, c));
568                         }
569                         bool dfs(int p, int f){
570                             if(!fc[f].use) return true;
571                             if(outside(p, f)){
572                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
573                                 fc[f].use=false;
574                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
575                                     ;
576                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
577                                     ;
578                                 return true;
579                             } else return false;
580                         }
581                         void build(){
582                             bool ok=false;
583                             fc.clear();
584                             sort(pt.begin(), pt.end(), point_cmp);
585                         }
586                     };
587                 };
588             };
589             template<typename T>
590             struct tetrahedron{ //四面體
591                 point3D<T> a,b,c,d;
592                 tetrahedron(){
593                     tetrahedron(const point3D<T> &a, const
594                         point3D<T> &b, const point3D<T> &c,
595                         const point3D<T> &d):a(a),b(b),c(c),d(
596                             d){
597                     T volume6(){const{ //體積的六倍
598                         return (d-a).dot((b-a).cross(c-a));
599                     }
600                     point3D<T> centroid(){const{
601                         return (a+b+c+d)/4;
602                     }
603                     bool point_in(const point3D<T> &p) const{
604                         return triangle3D<T>(a,b,c).point_in(p)
605                             &&triangle3D<T>(c,d,a).point_in(p);
606                     }
607                 };
608                 template<typename T>
609                 struct convexhull3D{
610                     static const int MAXN=105;
611                     struct face{
612                         int a,b,c;
613                         bool use;
614                         face(){
615                             face(int a, int b, int c):a(a),b(b),c(c),
616                                 use(1){
617                             };
618                         vector<point3D<T> > pt;
619                         vector<face> fc;
620                         int fid[MAXN][MAXN];
621                         static bool point_cmp(const point3D<T> &a,
622                             const point3D<T> &b){
623                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
624                                 y==b.y&&a.z<b.z)));
625                         }
626                         bool outside(int p, int a, int b, int c) const
627                             {
628                         return tetrahedron<T>(pt[a], pt[b], pt[c],
629                             pt[p]).volume6()<0;
630                         }
631                         bool outside(int p, int f) const{ return
632                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
633                         void add_face(int a, int b, int c, int p){
634                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
635                                 fid[a][c]=fc.size(), fc.push_back(
636                                     face(c, b, a));
637                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
638                                 size(), fc.push_back(face(a, b, c));
639                         }
640                         bool dfs(int p, int f){
641                             if(!fc[f].use) return true;
642                             if(outside(p, f)){
643                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
644                                 fc[f].use=false;
645                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
646                                     ;
647                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
648                                     ;
649                                 return true;
650                             } else return false;
651                         }
652                         void build(){
653                             bool ok=false;
654                             fc.clear();
655                             sort(pt.begin(), pt.end(), point_cmp);
656                         }
657                     };
658                 };
659             };
660             template<typename T>
661             struct tetrahedron{ //四面體
662                 point3D<T> a,b,c,d;
663                 tetrahedron(){
664                     tetrahedron(const point3D<T> &a, const
665                         point3D<T> &b, const point3D<T> &c,
666                         const point3D<T> &d):a(a),b(b),c(c),d(
667                             d){
668                     T volume6(){const{ //體積的六倍
669                         return (d-a).dot((b-a).cross(c-a));
670                     }
671                     point3D<T> centroid(){const{
672                         return (a+b+c+d)/4;
673                     }
674                     bool point_in(const point3D<T> &p) const{
675                         return triangle3D<T>(a,b,c).point_in(p)
676                             &&triangle3D<T>(c,d,a).point_in(p);
677                     }
678                 };
679                 template<typename T>
680                 struct convexhull3D{
681                     static const int MAXN=105;
682                     struct face{
683                         int a,b,c;
684                         bool use;
685                         face(){
686                             face(int a, int b, int c):a(a),b(b),c(c),
687                                 use(1){
688                             };
689                         vector<point3D<T> > pt;
690                         vector<face> fc;
691                         int fid[MAXN][MAXN];
692                         static bool point_cmp(const point3D<T> &a,
693                             const point3D<T> &b){
694                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
695                                 y==b.y&&a.z<b.z)));
696                         }
697                         bool outside(int p, int a, int b, int c) const
698                             {
699                         return tetrahedron<T>(pt[a], pt[b], pt[c],
700                             pt[p]).volume6()<0;
701                         }
702                         bool outside(int p, int f) const{ return
703                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
704                         void add_face(int a, int b, int c, int p){
705                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
706                                 fid[a][c]=fc.size(), fc.push_back(
707                                     face(c, b, a));
708                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
709                                 size(), fc.push_back(face(a, b, c));
710                         }
711                         bool dfs(int p, int f){
712                             if(!fc[f].use) return true;
713                             if(outside(p, f)){
714                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
715                                 fc[f].use=false;
716                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
717                                     ;
718                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
719                                     ;
720                                 return true;
721                             } else return false;
722                         }
723                         void build(){
724                             bool ok=false;
725                             fc.clear();
726                             sort(pt.begin(), pt.end(), point_cmp);
727                         }
728                     };
729                 };
730             };
731             template<typename T>
732             struct tetrahedron{ //四面體
733                 point3D<T> a,b,c,d;
734                 tetrahedron(){
735                     tetrahedron(const point3D<T> &a, const
736                         point3D<T> &b, const point3D<T> &c,
737                         const point3D<T> &d):a(a),b(b),c(c),d(
738                             d){
739                     T volume6(){const{ //體積的六倍
740                         return (d-a).dot((b-a).cross(c-a));
741                     }
742                     point3D<T> centroid(){const{
743                         return (a+b+c+d)/4;
744                     }
745                     bool point_in(const point3D<T> &p) const{
746                         return triangle3D<T>(a,b,c).point_in(p)
747                             &&triangle3D<T>(c,d,a).point_in(p);
748                     }
749                 };
750                 template<typename T>
751                 struct convexhull3D{
752                     static const int MAXN=105;
753                     struct face{
754                         int a,b,c;
755                         bool use;
756                         face(){
757                             face(int a, int b, int c):a(a),b(b),c(c),
758                                 use(1){
759                             };
760                         vector<point3D<T> > pt;
761                         vector<face> fc;
762                         int fid[MAXN][MAXN];
763                         static bool point_cmp(const point3D<T> &a,
764                             const point3D<T> &b){
765                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
766                                 y==b.y&&a.z<b.z)));
767                         }
768                         bool outside(int p, int a, int b, int c) const
769                             {
770                         return tetrahedron<T>(pt[a], pt[b], pt[c],
771                             pt[p]).volume6()<0;
772                         }
773                         bool outside(int p, int f) const{ return
774                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
775                         void add_face(int a, int b, int c, int p){
776                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
777                                 fid[a][c]=fc.size(), fc.push_back(
778                                     face(c, b, a));
779                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
780                                 size(), fc.push_back(face(a, b, c));
781                         }
782                         bool dfs(int p, int f){
783                             if(!fc[f].use) return true;
784                             if(outside(p, f)){
785                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
786                                 fc[f].use=false;
787                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
788                                     ;
789                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
790                                     ;
791                                 return true;
792                             } else return false;
793                         }
794                         void build(){
795                             bool ok=false;
796                             fc.clear();
797                             sort(pt.begin(), pt.end(), point_cmp);
798                         }
799                     };
800                 };
801             };
802             template<typename T>
803             struct tetrahedron{ //四面體
804                 point3D<T> a,b,c,d;
805                 tetrahedron(){
806                     tetrahedron(const point3D<T> &a, const
807                         point3D<T> &b, const point3D<T> &c,
808                         const point3D<T> &d):a(a),b(b),c(c),d(
809                             d){
810                     T volume6(){const{ //體積的六倍
811                         return (d-a).dot((b-a).cross(c-a));
812                     }
813                     point3D<T> centroid(){const{
814                         return (a+b+c+d)/4;
815                     }
816                     bool point_in(const point3D<T> &p) const{
817                         return triangle3D<T>(a,b,c).point_in(p)
818                             &&triangle3D<T>(c,d,a).point_in(p);
819                     }
820                 };
821                 template<typename T>
822                 struct convexhull3D{
823                     static const int MAXN=105;
824                     struct face{
825                         int a,b,c;
826                         bool use;
827                         face(){
828                             face(int a, int b, int c):a(a),b(b),c(c),
829                                 use(1){
830                             };
831                         vector<point3D<T> > pt;
832                         vector<face> fc;
833                         int fid[MAXN][MAXN];
834                         static bool point_cmp(const point3D<T> &a,
835                             const point3D<T> &b){
836                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
837                                 y==b.y&&a.z<b.z)));
838                         }
839                         bool outside(int p, int a, int b, int c) const
840                             {
841                         return tetrahedron<T>(pt[a], pt[b], pt[c],
842                             pt[p]).volume6()<0;
843                         }
844                         bool outside(int p, int f) const{ return
845                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
846                         void add_face(int a, int b, int c, int p){
847                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
848                                 fid[a][c]=fc.size(), fc.push_back(
849                                     face(c, b, a));
850                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
851                                 size(), fc.push_back(face(a, b, c));
852                         }
853                         bool dfs(int p, int f){
854                             if(!fc[f].use) return true;
855                             if(outside(p, f)){
856                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
857                                 fc[f].use=false;
858                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
859                                     ;
860                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
861                                     ;
862                                 return true;
863                             } else return false;
864                         }
865                         void build(){
866                             bool ok=false;
867                             fc.clear();
868                             sort(pt.begin(), pt.end(), point_cmp);
869                         }
870                     };
871                 };
872             };
873             template<typename T>
874             struct tetrahedron{ //四面體
875                 point3D<T> a,b,c,d;
876                 tetrahedron(){
877                     tetrahedron(const point3D<T> &a, const
878                         point3D<T> &b, const point3D<T> &c,
879                         const point3D<T> &d):a(a),b(b),c(c),d(
880                             d){
881                     T volume6(){const{ //體積的六倍
882                         return (d-a).dot((b-a).cross(c-a));
883                     }
884                     point3D<T> centroid(){const{
885                         return (a+b+c+d)/4;
886                     }
887                     bool point_in(const point3D<T> &p) const{
888                         return triangle3D<T>(a,b,c).point_in(p)
889                             &&triangle3D<T>(c,d,a).point_in(p);
890                     }
891                 };
892                 template<typename T>
893                 struct convexhull3D{
894                     static const int MAXN=105;
895                     struct face{
896                         int a,b,c;
897                         bool use;
898                         face(){
899                             face(int a, int b, int c):a(a),b(b),c(c),
900                                 use(1){
901                             };
902                         vector<point3D<T> > pt;
903                         vector<face> fc;
904                         int fid[MAXN][MAXN];
905                         static bool point_cmp(const point3D<T> &a,
906                             const point3D<T> &b){
907                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
908                                 y==b.y&&a.z<b.z)));
909                         }
910                         bool outside(int p, int a, int b, int c) const
911                             {
912                         return tetrahedron<T>(pt[a], pt[b], pt[c],
913                             pt[p]).volume6()<0;
914                         }
915                         bool outside(int p, int f) const{ return
916                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
917                         void add_face(int a, int b, int c, int p){
918                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
919                                 fid[a][c]=fc.size(), fc.push_back(
920                                     face(c, b, a));
921                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
922                                 size(), fc.push_back(face(a, b, c));
923                         }
924                         bool dfs(int p, int f){
925                             if(!fc[f].use) return true;
926                             if(outside(p, f)){
927                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
928                                 fc[f].use=false;
929                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
930                                     ;
931                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
932                                     ;
933                                 return true;
934                             } else return false;
935                         }
936                         void build(){
937                             bool ok=false;
938                             fc.clear();
939                             sort(pt.begin(), pt.end(), point_cmp);
940                         }
941                     };
942                 };
943             };
944             template<typename T>
945             struct tetrahedron{ //四面體
946                 point3D<T> a,b,c,d;
947                 tetrahedron(){
948                     tetrahedron(const point3D<T> &a, const
949                         point3D<T> &b, const point3D<T> &c,
950                         const point3D<T> &d):a(a),b(b),c(c),d(
951                             d){
952                     T volume6(){const{ //體積的六倍
953                         return (d-a).dot((b-a).cross(c-a));
954                     }
955                     point3D<T> centroid(){const{
956                         return (a+b+c+d)/4;
957                     }
958                     bool point_in(const point3D<T> &p) const{
959                         return triangle3D<T>(a,b,c).point_in(p)
960                             &&triangle3D<T>(c,d,a).point_in(p);
961                     }
962                 };
963                 template<typename T>
964                 struct convexhull3D{
965                     static const int MAXN=105;
966                     struct face{
967                         int a,b,c;
968                         bool use;
969                         face(){
970                             face(int a, int b, int c):a(a),b(b),c(c),
971                                 use(1){
972                             };
973                         vector<point3D<T> > pt;
974                         vector<face> fc;
975                         int fid[MAXN][MAXN];
976                         static bool point_cmp(const point3D<T> &a,
977                             const point3D<T> &b){
978                             return a.x<b.x|| (a.x==b.x&&(a.y<b.y|| (a.
979                                 y==b.y&&a.z<b.z)));
980                         }
981                         bool outside(int p, int a, int b, int c) const
982                             {
983                         return tetrahedron<T>(pt[a], pt[b], pt[c],
984                             pt[p]).volume6()<0;
985                         }
986                         bool outside(int p, int f) const{ return
987                             outside(p, fc[f].a, fc[f].b, fc[f].c); }
988                         void add_face(int a, int b, int c, int p){
989                             if(outside(p, a, b, c)) fid[c][b]=fid[b][a]=
990                                 fid[a][c]=fc.size(), fc.push_back(
991                                     face(c, b, a));
992                             else fid[a][b]=fid[b][c]=fid[c][a]=fc.
993                                 size(), fc.push_back(face(a, b, c));
994                         }
995                         bool dfs(int p, int f){
996                             if(!fc[f].use) return true;
997                             if(outside(p, f)){
998                                 int a=fc[f].a, b=fc[f].b, c=fc[f].c;
999                                 fc[f].use=false;
1000                                 if(!dfs(p, fid[b][a])) add_face(p, a, b, c)
1001                                     ;
1002                                 if(!dfs(p, fid[a][c])) add_face(p, c, a, b)
1003                                     ;
1004                                 return true;
1005                             } else return false;
1006                         }
1007                         void build(){
1008                             bool ok=false;
1009                             fc.clear();
1010                             sort(pt.begin(), pt.end(), point_cmp);
1011                         }
1012                     };
1013                 };
1014             };
1015             template<typename T>
1016             struct tetrahedron{ //四面體
1017                 point3D<T> a,b,c,d;
1018                 tetrahedron(){
1019                     tetrahedron(const point3D<T> &a, const
1020                         point3D<T> &b, const point3D<T> &c,
1021                         const point3D<T> &d):a(a),b(b),c(c),d(
1022                             d){
1023                     T volume6(){const{ //體積的六倍
10
```

```

392 pt.resize(unique(pt.begin(),pt.end())-pt
    .begin());
393 for(size_t i=2;i<pt.size();++i){
394     if((pt[0]-pt[i]).area2(pt[1]-pt[i])
        !=0){
395         ok=true;
396         swap(pt[i],pt[2]);
397         break;
398     }
399 }
400 if(!ok)return;
401 ok=false;
402 for(size_t i=3;i<pt.size();++i){
403     if(tetrahedron<T>(pt[0],pt[1],pt[2],pt
        [i]).volume6()!=0){
404         ok=true;
405         swap(pt[i],pt[3]);
406         break;
407     }
408 }
409 if(!ok)return;
410 for(int i=0;i<4;++i)add_face(i,(i+1)%4,(
    i+2)%4,(i+3)%4);
411 for(size_t i=4;i<pt.size();++i){
412     for(int j=fc.size()-1;j>0;--j){
413         if(outside(i,j)){
414             dfs(i,j);
415             break;
416         }
417     }
418 }
419 size_t sz=0;
420 for(size_t i=0;i<fc.size();++i)if(fc[i].
    use)fc[sz++]=fc[i];
421 fc.resize(sz);
422 }
423 point3D<T> centroid()const{
424     point3D<T> res(0,0,0);
425     T vol=0;
426     for(size_t i=0;i<fc.size();++i){
427         T tmp=pt[fc[i].a].dot(pt[fc[i].b].
            cross(pt[fc[i].c]));
428         res=res+(pt[fc[i].a]+pt[fc[i].b]+pt[fc
            [i].c])*tmp;
429         vol+=tmp;
430     }
431     return res/(vol*4);
432 }
433 };

```

1.3 SmallestCircle.cpp

```

1 #include "Geometry.cpp"
2 #include <vector>
3 struct Circle{
4     typedef point<double> p;
5     typedef const point<double> cp;
6     p x;
7     double r2;
8     bool incircle(cp &c)const{return (x-c).
        abs2()<=r2;};
9 };
10

```

```

11 Circle TwoPointCircle(Circle::cp &a, Circle
    ::cp &b) {
12     Circle::p m=(a+b)/2;
13     return Circle{m,(a-m).abs2()};
14 }
15
16 Circle outcircle(Circle::p a, Circle::p b,
    Circle::p c) {
17     if(TwoPointCircle(a,b).incircle(c))
        return TwoPointCircle(a,b);
18     if(TwoPointCircle(b,c).incircle(a))
        return TwoPointCircle(b,c);
19     if(TwoPointCircle(c,a).incircle(b))
        return TwoPointCircle(c,a);
20     Circle::p ret;
21     double a1=b.x-a.x, b1=b.y-a.y, c1=(a1*a1
        +b1*b1)/2;
22     double a2=c.x-a.x, b2=c.y-a.y, c2=(a2*a2
        +b2*b2)/2;
23     double d = a1*b2 - a2*b1;
24     ret.x=a.x+(c1*b2-c2*b1)/d;
25     ret.y=a.y+(a1*c2-a2*c1)/d;
26     return Circle{ret,(ret-a).abs2()};
27 }
28 //rand required
29 Circle SmallestCircle(std::vector<Circle::p>
    &p){
30     int n=p.size();
31     if(n==1) return Circle{p[0],0,0};
32     if(n==2) return TwoPointCircle(p[0],p
        [1]);
33     random_shuffle(p.begin(),p.end());
34     Circle c = {p[0],0,0};
35     for(int i=0;i<n;++i){
36         if(c.incircle(p[i])) continue;
37         c=Circle{p[i],0,0};
38         for(int j=0;j<i;++j){
39             if(c.incircle(p[j])) continue;
40             c=TwoPointCircle(p[i],p[j]);
41             for(int k=0;k<j;++k){
42                 if(c.incircle(p[k]))
                    continue;
43                 c=outcircle(p[i],p[j],p[k]);
44             }
45         }
46     }
47     return c;
48 }

```

1.4 最近點對.cpp

```

1 #define INF LLONG_MAX/*預設是Long Long最大值
    */
2 template<typename T>
3 T closest_pair(vector<point<T>> &v,vector<
    point<T>> &t,int l,int r){
4     T dis=INF,tmd;
5     if(l==r)return dis;
6     int mid=(l+r)/2;
7     if((tmd=closest_pair(v,t,l,mid))<dis)dis=
        tmd;
8     if((tmd=closest_pair(v,t,mid+1,r))<dis)dis
        =tmd;

```

```

9     t.clear();
10    for(int i=l;i<=r;++i)
11        if((v[i].x-v[mid].x)*(v[i].x-v[mid].x)<
            dis)t.push_back(v[i]);
12    sort(t.begin(),t.end(),point<T>::y_cmp);/*
        如果用merge_sort的方式可以O(n)*
13    for(int i=0;i<(int)t.size();++i)
14        for(int j=1;j<=3&&i+j<(int)t.size();++j)
15            if((tmd=(t[i]-t[i+j]).abs2())<dis)dis=
                tmd;
16    return dis;
17 }
18 template<typename T>
19 inline T closest_pair(vector<point<T>> &v){
20     vector<point<T>> t;
21     sort(v.begin(),v.end(),point<T>::x_cmp);
22     return closest_pair(v,t,0,v.size()-1);/*最
        近點對距離*/
23 }

```

1.5 浮點數誤差模板.cpp

```

1 const double EPS=1e-9;
2 struct Double{
3     double d;
4     Double(double d=0):d(d){}
5     bool operator <(const Double &b)const{
6         return d-b.d<-EPS;};
7     bool operator >(const Double &b)const{
8         return d-b.d>EPS;};
9     bool operator ==(const Double &b)const{
10        return fabs(d-b.d)<=EPS;};
11    bool operator !=(const Double &b)const{
12        return fabs(d-b.d)>EPS;};
13    bool operator <=(const Double &b)const{
14        return d-b.d<=EPS;};
15    bool operator >=(const Double &b)const{
16        return d-b.d>=-EPS;};
17    operator double()const{return d;};
18 };

```

2 Data_Structure

2.1 DLX.cpp

```

1 #define MAXN 4100
2 #define MAXM 1030
3 #define MAXND 16390
4 struct DLX{
5     int n,m,sz,ansd;//高是n·寬是m的稀疏矩陣
6     int S[MAXN],H[MAXN];
7     int row[MAXN],col[MAXND];/*每個節點代表的
        列跟行
8     int L[MAXND],R[MAXND],U[MAXND],D[MAXND];
9     vector<int> ans,anst;
10    void init(int _n,int _m){
11        n=_n,m=_m;

```

```

12    for(int i=0;i<=m;++i){
13        U[i]=D[i]=i,L[i]=i-1,R[i]=i+1;
14        S[i]=0;
15    }
16    R[m]=0,L[0]=m;
17    sz=m,ansd=INT_MAX;/*ansd存最優解的個數
18    for(int i=1;i<=n;++i)H[i]=-1;
19 }
20 void add(int r,int c){
21     ++S[col[ansd]=c];
22     row[sz]=r;
23     D[sz]=D[c],U[D[c]]=sz,U[sz]=c,D[c]=sz;
24     if(H[r]<0)H[r]=L[sz]=R[sz]=sz;
25     else R[sz]=R[H[r]],L[R[H[r]]]=sz,L[sz]=H
        [r],R[H[r]]=sz;
26 }
27 #define DFOR(i,A,s) for(int i=A[s];i!=s;i=
    A[i])
28 void remove(int c){//刪除第c行和所有當前覆
    蓋到第c行的列
29     L[R[c]]=L[c],R[L[c]]=R[c];/*這裡刪除第c
        行·若有些行不需要處理可以在開始時呼
        叫他
30     DFOR(i,D,c)DFOR(j,R,i){U[D[j]]=U[j],D[U
        [j]]=D[j],--S[col[j]]};
31 }
32 void restore(int c){//恢復第c行和所有當前
    覆蓋到第c行的列·remove的逆操作
33     DFOR(i,U,c)DFOR(j,L,i){++S[col[j]],U[D[j
        ]]=j,D[U[j]]=j;};
34     L[R[c]]=c,R[L[c]]=c;
35 }
36 void remove2(int nd){//刪除nd所在的行當前
    所有點(包括虛擬節點)·只保留nd
37     DFOR(i,D,nd)L[R[i]]=L[i],R[L[i]]=R[i];
38 }
39 void restore2(int nd){//刪除nd所在的行當前
    所有點·為remove2的逆操作
40     DFOR(i,U,nd)L[R[i]]=R[L[i]]=i;
41 }
42 bool vis[MAXN];
43 int h();//估價函數 for IDA*
44 int res=0;
45 memset(vis,0,sizeof(vis));
46 DFOR(i,R,0)if(!vis[i]){
47     vis[i]=1;
48     ++res;
49     DFOR(j,D,i)DFOR(k,R,j)vis[col[k]]=1;
50 }
51 return res;
52 }
53 bool dfs(int d){//for精確覆蓋問題
54     if(d+h()>=ansd)return 0;/*找最佳解用·找
        任意解可以刪掉
55     if(!R[0]){ansd=d;return 1;};
56     int c=R[0];
57     DFOR(i,R,0)if(S[i]<S[c])c=i;
58     remove(c);
59     DFOR(i,D,c){
60         ans.push_back(row[i]);
61         DFOR(j,R,i)remove(col[j]);
62         if(dfs(d+1))return 1;
63         ans.pop_back();

```

```

143         nearest(u->l, (k+1)%kd, x, h, mndist);
144     }
145     h[k]=old;
146 }
147 std::vector<point> in_range;
148 void range(node *u, int k, const point&mi,
149           const point&ma){
150     if(!u) return;
151     bool is=1;
152     for(int i=0; i<kd; ++i)
153         if(u->pid.d[i]<mi.d[i] || ma.d[i]<u->
154            pid.d[i]){
155             is=0; break;
156         }
157     if(is) in_range.push_back(u->pid);
158     if(mi.d[k]<=u->pid.d[k]) range(u->l, (k
159                                     +1)%kd, mi, ma);
160     if(ma.d[k]>=u->pid.d[k]) range(u->r, (k
161                                     +1)%kd, mi, ma);
162 }
163 public:
164 kd_tree(const T &INF, double a=0.75):root(
165     (0), alpha(a), loga(log2(1.0/a)), INF(
166     INF), maxn(1)){
167     ~kd_tree(){ delete root; }
168     void clear(){ delete root, root=0, maxn=1; }
169     void build(int n, const point *p){
170         delete root, A.resize(maxn=n);
171         for(int i=0; i<n; ++i) A[i]=new node(p[i
172             ]);
173         root=build(0, n-1);
174     }
175     void insert(const point &x){
176         insert(root, 0, x, __lg(size(root))/loga)
177         ;
178         if(root->s>maxn) maxn=root->s;
179     }
180     bool erase(const point &p){
181         bool d=erase(root, 0, p);
182         if(root&&root->s<alpha*maxn) rebuild();
183         return d;
184     }
185     void rebuild(){
186         if(root) rebuild(root, 0);
187         maxn=root->s;
188     }
189     T nearest(const point &x, int k){
190         qM=k;
191         T mndist=INF, h[kd]={};
192         nearest(root, 0, x, h, mndist);
193         mndist=pQ.top().first;
194         pQ=std::priority_queue<std::pair<T,
195             point >>());
196         return mndist; /*!^9000x^20k000I009Z00*/
197     }
198     const std::vector<point> &range(const
199         point&mi, const point&ma){
200         in_range.clear();
201         range(root, 0, mi, ma);
202         return in_range; /*!^9G900mi^0ma059j;000
203             Ivector*/
204     }
205     int size(){ return root?root->s:0; }
206 };

```


2.3 kd_tree_replace_segment_tree

```

1  /*kd樹代替高維線段樹*/
2  struct node{
3      node *l,*r;
4      point pid,mi,ma;
5      int s;
6      int data;
7      node(const point &p,int d):l(0),r(0),pid(p),mi(p),ma(p),s(1),data(d),dmin(d),dmax(d){}
8      void up(){
9          mi=ma=pid;
10         s=1;
11         if(l){
12             for(int i=0;i<kd;++i){
13                 mi.d[i]=min(mi.d[i],l->mi.d[i]);
14                 ma.d[i]=max(ma.d[i],l->ma.d[i]);
15             }
16             s+=l->s;
17         }
18         if(r){
19             for(int i=0;i<kd;++i){
20                 mi.d[i]=min(mi.d[i],r->mi.d[i]);
21                 ma.d[i]=max(ma.d[i],r->ma.d[i]);
22             }
23             s+=r->s;
24         }
25     }
26     void up2(){
27         //其他懶惰標記向上更新
28     }
29     void down(){
30         //其他懶惰標記下推
31     }
32 }*root;
33
34 /*檢查區間包含用的函數*/
35 inline bool range_include(node *o,const point &L,const point &R){
36     for(int i=0;i<kd;++i){
37         if(L.d[i]>o->ma.d[i]||R.d[i]<o->mi.d[i])
38             return 0;
39     }
40     //只要(L,R)區間有和o的區間有交集就回傳true
41     return 1;
42 }
43 inline bool range_in_range(node *o,const point &L,const point &R){
44     for(int i=0;i<kd;++i){
45         if(L.d[i]>o->mi.d[i]||o->ma.d[i]>R.d[i])
46             return 0;
47     }
48     //如果(L,R)區間完全包含o的區間就回傳true
49     return 1;
50 }
51 inline bool point_in_range(node *o,const point &L,const point &R){
52     for(int i=0;i<kd;++i){
53         if(L.d[i]>o->pid.d[i]||R.d[i]<o->pid.d[i])
54             return 0;
55     }
56     //如果(L,R)區間完全包含o->pid這個點就回傳true
57     return 1;
58 }
59
60 /*單點修改·以單點改值為例*/
61 void update(node *u,const point &x,int data,int k=0){
62     if(!u)return;
63     u->down();
64     if(u->pid==x){
65         u->data=data;
66         u->up2();
67         return;
68     }
69     cmp.sort_id=k;
70     update(cmp(x,u->pid)?u->l:u->r,x,data,(k+1)%kd);
71     u->up2();
72 }
73
74 /*區間修改*/
75 void update(node *o,const point &L,const point &R,int data){
76     if(!o)return;
77     o->down();
78     if(range_in_range(o,L,R)){
79         //區間懶惰標記修改
80         o->down();
81         return;
82     }
83     if(point_in_range(o,L,R)){
84         //這個點在(L,R)區間·但是他的左右子樹不一定在區間中
85         //單點懶惰標記修改
86     }
87     if(o->l&&range_include(o->l,L,R))update(o->l,L,R,data);
88     if(o->r&&range_include(o->r,L,R))update(o->r,L,R,data);
89     o->up2();
90 }
91
92 /*區間查詢·以總和為例*/
93 int query(node *o,const point &L,const point &R){
94     if(!o)return 0;
95     o->down();
96     if(range_in_range(o,L,R))return o->sum;
97     int ans=0;
98     if(point_in_range(o,L,R))ans+=o->data;
99     if(o->l&&range_include(o->l,L,R))ans+=query(o->l,L,R);
100    if(o->r&&range_include(o->r,L,R))ans+=query(o->r,L,R);
101    return ans;
102 }

```

2.4 persistent_segment_tree.cpp

```

1  #include<bits/stdc++.h>//POJ 2104
2  using namespace std;
3  struct node{
4      int l,r;
5      int data;

```

```

6      node(int l,int r,int d):l(l),r(r),data(d){}
7  };
8  vector<node> nds;
9  inline void up(int o,int l,int r){
10     nds[o].data=nds[l].data+nds[r].data;
11 }
12 inline int new_node(int l,int r,int d){
13     nds.push_back(node(l,r,d));
14     return nds.size()-1;
15 }
16 inline int new_node(const node &nd){
17     nds.push_back(nd);
18     return nds.size()-1;
19 }
20 int build_tree(int l,int r){
21     int nd=new_node(-1,-1,0);
22     if(l==r)return nd;
23     int mid=(l+r)/2;
24     int L=build_tree(l,mid);//執行時vector會被重構
25     int R=build_tree(mid+1,r);//一定要這樣寫
26     nds[nd].l=L;
27     nds[nd].r=R;
28     //up(nd,L,R);
29     return nd;
30 }
31 int insert(int l,int r,int rt,int x,int d){
32     if(x<l||r<x)return rt;
33     int nd=new_node(nds[rt]);
34     if(l==r&&l==x)nds[nd].data+=d;
35     else{
36         int mid=(l+r)/2;
37         int L=insert(l,mid,nds[nd].l,x,d);
38         int R=insert(mid+1,r,nds[nd].r,x,d);
39         nds[nd].l=L;
40         nds[nd].r=R;
41         up(nd,L,R);
42     }
43     return nd;
44 }
45 inline int cal(int L,int R){
46     return nds[R].data-nds[L].data;
47 }
48 int find(int l,int r,int L,int R,int k){
49     if(l==r)return l;
50     int mid=(l+r)/2;
51     int add=cal(nds[L].l,nds[R].l);
52     if(k<=add)return find(l,mid,nds[L].l,nds[R].l,k);
53     return find(mid+1,r,nds[L].r,nds[R].r,k-add);
54 }
55 int n,m;
56 int s[100005];
57 int root[100005];
58 int main(){
59     while(~scanf("%d",&n,&m)){
60         nds.clear();
61         vector<int> lsh;
62         for(int i=1;i<=n;++i){
63             scanf("%d",&s[i]);
64             lsh.push_back(s[i]);
65         }
66         sort(lsh.begin(),lsh.end());

```

```

67         lsh.resize(unique(lsh.begin(),lsh.end())-lsh.begin());
68         int N=(int)lsh.size()-1;
69         root[0]=build_tree(0,N);
70         for(int i=1;i<=m;++i){
71             s[i]=lower_bound(lsh.begin(),lsh.end(),s[i])-lsh.begin();
72             root[i]=insert(0,N,root[i-1],s[i],1);
73         }
74         while(m--){
75             int a,b,k;
76             scanf("%d%d",&a,&b,&k);
77             int res=find(0,N,root[a-1],root[b],k);
78             printf("%d\n",lsh[res]);
79         }
80     }
81     return 0;
82 }

```

2.5 reference_point.cpp

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  template<typename T>
4  struct _RefCounter{
5      T data;
6      int ref;
7      _RefCounter(const T&d=0):data(d),ref(0){}
8  };
9  template<typename T>
10 struct reference_pointer{
11     _RefCounter<T> *p;
12     T *operator->(){return &(*p).data;}
13     T &operator*(){return *p->data;}
14     operator int(){return (int)(long long)p;}
15     reference_pointer&operator=(const reference_pointer &t){
16         if(p&&--(*p).ref==0)delete p;
17         p=t.p;
18         p&&+(*p).ref;
19         return *this;
20     }
21     reference_pointer(_RefCounter<T> *t=0):p(t){}
22     p&&+(*p).ref;
23 }
24 reference_pointer(const reference_pointer &t):p(t.p){
25     p&&+(*p).ref;
26 }
27 ~reference_pointer(){
28     if(p&&--(*p).ref==0)delete p;
29 }
30 };
31 template<typename T>
32 inline const reference_pointer<T> new_reference(const T&nd){
33     return reference_pointer<T>(new _RefCounter<T>(nd));
34 }
35 struct P{
36     int a,b;
37     P(int A,int B):a(A),b(B){}

```

```

38 }p(2,3);
39 int main(){
40     reference_pointer<int >b=new_reference(int
41         (5));
42     reference_pointer<int >a=new_reference(*b)
43         ;
44     reference_pointer<P >c=new_reference(p);
45     return 0;
46 }

```

2.6 skew_heap.cpp

```

1 template<typename T,typename _Compare=std::
2     less<T> >
3 class skew_heap{
4     private:
5         struct node{
6             T data;
7             node *l,*r;
8             node(const T&d):data(d),l(0),r(0){}
9             ~node(){delete l,delete r;}
10        }*root;
11        int _size;
12        _Compare cmp;
13        node *merge(node *a,node *b){
14            if(!a||!b)return a?a:b;
15            if(cmp(a->data,b->data))return merge(b,a);
16            node *t=a->r;
17            a->r=a->l;
18            a->l=merge(b,t);
19            return a;
20        }
21        public:
22        skew_heap():root(0),_size(0){}
23        ~skew_heap(){delete root;}
24        void clear(){delete root,root=0,_size=0;}
25        void join(skew_heap &o){
26            root=merge(root,o.root);
27            o.root=0;
28            _size+=o._size;
29            o._size=0;
30        }
31        void swap(skew_heap &o){
32            node *t=root;
33            root=o.root;
34            o.root=t;
35            int st=_size;
36            _size=o._size;
37            o._size=st;
38        }
39        void push(const T&data){
40            _size++;
41            root=merge(root,new node(data));
42        }
43        void pop(){
44            if(_size)_size--;
45            node *tmd=merge(root->l,root->r);
46            root->l=root->r=0;
47            delete root;
48            root=tmd;
49        }

```

2.7 split_merge.cpp

```

49 const T& top(){return root->data;}
50 int size(){return _size;}
51 bool empty(){return !_size;}
52 };
2 void split(node *o,node *a,node *b,int k){
3     if(!o)a=b=0;
4     else{
5         //o=new node(*o);
6         o->down();
7         if(k<=size(o->l)){
8             b=0;
9             split(o->l,a,b->l,k);
10        }else{
11            a=0;
12            split(o->r,a->r,b,k-size(o->l)-1);
13        }
14        o->up();
15    }
16    node *merge(node *a,node *b){
17        if(!a||!b)return a?a:b;
18        static int x;
19        if(x++%(a->s+b->s)<a->s){
20            //a=new node(*a);
21            a->down();
22            a->r=merge(a->r,b);
23            a->up();
24            return a;
25        }else{
26            //b=new node(*b);
27            b->down();
28            b->l=merge(a,b->l);
29            b->up();
30            return b;
31        }
32    }

```

2.8 treap.cpp

```

1 template<typename T>
2 class treap{
3     private:
4         struct node{
5             T data;
6             unsigned fix;
7             int s;
8             node *ch[2];
9             node(const T&d):data(d),s(1){}
10            node():s(0){ch[0]=ch[1]=this;}
11        }*nil,*root;
12        unsigned x;
13        unsigned ran(){return x=x*0xdefaced+1;}
14        void rotate(node *a,bool d){
15            node *b=a;
16            a=a->ch[!d];
17            a->s=b->s;
18            b->ch[!d]=a->ch[d];

```

```

19        a->ch[d]=b;
20        b->s=b->ch[0]->s+b->ch[1]->s+1;
21    }
22    void insert(node *o,const T &data){
23        if(!o->s){
24            o=new node(data),o->fix=ran();
25            o->ch[0]=o->ch[1]=nil;
26        }else{
27            o->s++;
28            bool d=o->data<data;
29            insert(o->ch[d],data);
30            if(o->ch[d]->fix>o->fix)rotate(o,!d);
31        }
32    }
33    node *merge(node *a,node *b){
34        if(!a->s||!b->s)return a->s?a:b;
35        if(a->fix>b->fix){
36            a->ch[1]=merge(a->ch[1],b);
37            a->s=a->ch[0]->s+a->ch[1]->s+1;
38            return a;
39        }else{
40            b->ch[0]=merge(a,b->ch[0]);
41            b->s=b->ch[0]->s+b->ch[1]->s+1;
42            return b;
43        }
44    }
45    bool erase(node *o,const T &data){
46        if(!o->s)return 0;
47        if(o->data==data){
48            node *t=o;
49            o=merge(o->ch[0],o->ch[1]);
50            delete t;
51            return 1;
52        }
53        if(erase(o->ch[o->data<data],data)){
54            o->s--;return 1;
55        }else return 0;
56    }
57    void clear(node *o){
58        if(o->s)clear(o->ch[0]),clear(o->ch[1]),delete o;
59    }
60    public:
61    treap(unsigned s=20150119):nil(new node(s),root(nil),x(s){}){
62        ~treap(){clear(root),delete nil;}
63        void clear(){clear(root),root=nil;}
64        void insert(const T &data){
65            insert(root,data);
66        }
67        bool erase(const T &data){
68            return erase(root,data);
69        }
70        bool find(const T&data){
71            for(node *o=root;o->s;){
72                if(o->data==data)return 1;
73                else o=o->ch[o->data<data];
74            }
75            return 0;
76        }
77        int rank(const T&data){
78            int cnt=0;
79            for(node *o=root;o->s;){
80                if(o->data<data)cnt+=o->ch[0]->s+1,o=o->ch[1];
81                else o=o->ch[0];
82            }
83            return cnt;
84        }
85        const T&kth(int k){
86            for(node *o=root;){
87                if(k<=o->ch[0]->s)o=o->ch[0];
88                else if(k==o->ch[0]->s+1)return o->data;
89                else k=k-o->ch[0]->s+1,o=o->ch[1];
90            }
91        }
92        const T&operator[](int k){
93            return kth(k);
94        }
95        const T&preorder(const T&data){
96            node *x=root,*y=0;
97            while(x->s){
98                if(x->data<data)y=x,x=x->ch[1];
99                else x=x->ch[0];
100            }
101            if(y)return y->data;
102            return data;
103        }
104        const T&successor(const T&data){
105            node *x=root,*y=0;
106            while(x->s){
107                if(data<x->data)y=x,x=x->ch[0];
108                else x=x->ch[1];
109            }
110            if(y)return y->data;
111            return data;
112        }
113        int size(){return root->s;}
114    };

```

2.9 操作分治.cpp

```

1 void dq(int l,int r){
2     if(l==r)return;
3     int mid=(l+r)/2;
4     dq(l,mid);
5     //處理[l,mid]的操作對[mid+1,r]的影響
6     dq(mid+1,r);
7 }

```

2.10 整體二分.cpp

```

1 void BS(int l,int r,vector<Item> &vs){
2     //答案該<l會有的已經做完了
3     if(l==r)整個vs的答案=1;//??????
4     int mid=(l+r)/2;
5     do_thing(l,mid);//做答案<=mid會做的事
6     vector<Item> left=vs裡滿足的;
7     vector<Item> right=vs-left;
8     undo_thing(l,mid);
9     BS(l,mid,left);
10    do_thing(l,mid);
11    BS(mid+1,r,right);//??????
12 }

```

3 default

3.1 debug.cpp

```
1 #ifndef Jinkala
2 #define debug(...) {\
3     fprintf(stderr, "%s - %d : (%s) = ",
4         __PRETTY_FUNCTION__, __LINE__, #
5         __VA_ARGS__); \
6     _DO(__VA_ARGS__); \
7 }
8 template<typename I> void _DO(I&&x){cerr<<x
9     <<endl;}
10 template<typename I, typename... T> void _DO(I
11     &&x, T&&...tail){cerr<<x<<" "; _DO(tail
12     ...);}
13 #else
14 #define debug(...)
15 #endif
```

3.2 IncStack.cpp

```
1 //Magic
2 #pragma GCC optimize "Ofast"
3 //stack resize, change esp to rsp if 64-bit
4 //system
5 asm("mov %0, %%esp\n" :: "g"(mem+1000000));
6 //linux stack resize
7 #include<sys/resource.h>
8 void increase_stack(){
9     const rlim_t ks=64*1024*1024;
10     struct rlimit rl;
11     int res=getrlimit(RLIMIT_STACK, &rl);
12     if(!res&&rl.rlim_cur<ks){
13         rl.rlim_cur=ks;
14         res=setrlimit(RLIMIT_STACK, &rl);
15     }
16 }
```

3.3 input.cpp

```
1 inline int read(){
2     int x=0; bool f=0; char c=getchar();
3     while(ch<'0' || '9'<ch) f=(ch=='-'), ch=
4         getchar();
5     while('0'<=ch&&ch<='9') x=x*10-'0'+ch, ch=
6         getchar();
7     return f?-x:x;
8 }
9 //g++ -std=c++11 -O2 -Wall -Wextra -Wno-
10 unused-variable $1 && ./a.out
```

4 Flow

4.1 dinic.cpp

```
1 #define MAXN 105
2 #define INF INT_MAX
3 int n; /*number of nodes*/
4 int level[MAXN], cur[MAXN]; /*Layer, current
5     arc*/
6 struct edge{
7     int v, pre;
8     long long cap, flow, r;
9     edge(int v, int pre, long long cap):v(v), pre
10         (pre), cap(cap), flow(0), r(cap){}
11 };
12 int g[MAXN];
13 std::vector<edge> e;
14 inline void init(){
15     memset(g, -1, sizeof(int)*(n+1));
16     e.clear();
17 }
18 inline void add_edge(int u, int v, long long
19     cap, bool directed=false){
20     e.push_back(edge(v, g[u], cap));
21     g[u]=e.size()-1;
22     e.push_back(edge(u, g[v], directed?0:cap));
23     g[v]=e.size()-1;
24 }
25 inline int bfs(int s, int t){
26     memset(level, 0, sizeof(int)*(n+1));
27     memcpy(cur, g, sizeof(int)*(n+1));
28     std::queue<int> q;
29     q.push(s);
30     level[s]=1;
31     while(q.size()){
32         int u=q.front(); q.pop();
33         for(int i=g[u]; ~i; i=e[i].pre){
34             if(!level[e[i].v]&&e[i].r){
35                 level[e[i].v]=level[u]+1;
36                 q.push(e[i].v);
37                 if(e[i].v==t) return 1;
38             }
39         }
40     }
41     return 0;
42 }
43 long long dfs(int u, int t, long long cur_flow
44     =INF){
45     if(u==t || !cur_flow) return cur_flow;
46     long long df, tf=0;
47     for(int &i=cur[u]; ~i; i=e[i].pre){
48         if(level[e[i].v]==level[u]+1&&e[i].r){
49             if(df=dfs(e[i].v, t, std::min(cur_flow, e
50                 [i].r))) {
51                 e[i].flow+=df;
52                 e[i^1].flow-=df;
53                 e[i].r-=df;
54                 e[i^1].r+=df;
55                 tf+=df;
56                 if(!cur_flow==df) break;
57             }
58         }
59     }
60     return tf;
61 }
```

```
55 if(!df) level[u]=0;
56 return tf;
57 }
58 inline long long dinic(int s, int t, bool
59     clean=true){
60     if(clean){
61         for(size_t i=0; i<e.size(); ++i){
62             e[i].flow=0;
63             e[i].r=e[i].cap;
64         }
65     }
66     long long ans=0;
67     while(bfs(s, t)) ans+=dfs(s, t);
68     return ans;
69 }
```

4.2 ISAP.cpp

```
1 #define MAXN 105
2 #define INF INT_MAX
3 int n; /*點數*/
4 int d[MAXN], gap[MAXN], cur[MAXN];
5 /*層次、gap[i]=層次為i的點之個數、當前弧優化
6     */
7 struct edge{
8     int v, pre;
9     long long cap, flow, r;
10     edge(int v, int pre, long long cap):v(v), pre
11         (pre), cap(cap), flow(0), r(cap){}
12 };
13 int g[MAXN];
14 std::vector<edge> e;
15 inline void init(){
16     memset(g, -1, sizeof(int)*(n+1));
17     e.clear();
18 }
19 inline void add_edge(int u, int v, long long
20     cap, bool directed=false){
21     e.push_back(edge(v, g[u], cap));
22     g[u]=e.size()-1;
23     e.push_back(edge(u, g[v], directed?0:cap));
24     g[v]=e.size()-1;
25 }
26 long long dfs(int u, int s, int t, long long
27     cur_flow=INF){
28     if(u==t) return cur_flow;
29     long long tf=cur_flow, df;
30     for(int &i=cur[u]; ~i; i=e[i].pre){
31         if(e[i].r&&d[u]==d[e[i].v]+1){
32             df=dfs(e[i].v, s, t, std::min(tf, e[i].r))
33                 ;
34             e[i].flow+=df;
35             e[i^1].flow-=df;
36             e[i].r-=df;
37             e[i^1].r+=df;
38             if(!(tf-=df) || d[s]==n) return cur_flow-
39                 tf;
40         }
41     }
42     int minh=n;
43     for(int i=cur[u]=g[u]; ~i; i=e[i].pre){
44         if(e[i].r&&d[e[i].v]<minh) minh=d[e[i].v]
45             ;
46     }
```

```
39 }
40 if(!--gap[d[u]]) d[s]=n;
41 else ++gap[d[u]]=++minh;
42 return cur_flow-tf;
43 }
44 inline long long isap(int s, int t, bool clean
45     =true){
46     memset(d, 0, sizeof(int)*(n+1));
47     memset(gap, 0, sizeof(int)*(n+1));
48     memcpy(cur, g, sizeof(int)*(n+1));
49     if(clean){
50         for(size_t i=0; i<e.size(); ++i){
51             e[i].flow=0;
52             e[i].r=e[i].cap;
53         }
54     }
55     long long max_flow=0;
56     for(gap[0]=n; d[s]<n; ) max_flow+=dfs(s, s, t);
57     return max_flow;
58 }
```

4.3 MinCostMaxFlow.cpp

```
1 #define MAXN 440
2 #define INF 999999999
3 struct edge{
4     int v, pre;
5     int cap, cost;
6     edge(int v, int pre, int cap, int cost):v(v),
7         pre(pre), cap(cap), cost(cost){}
8 };
9 int n, S, T;
10 int dis[MAXN], piS, ans;
11 bool vis[MAXN];
12 std::vector<edge> e;
13 int g[MAXN];
14 inline void init(){
15     memset(g, -1, sizeof(int)*n);
16     e.clear();
17 }
18 inline void add_edge(int u, int v, int cost,
19     int cap, bool directed=false){
20     e.push_back(edge(v, g[u], cap, cost));
21     g[u]=e.size()-1;
22     e.push_back(edge(u, g[v], directed?0:cap, -
23         cost));
24     g[v]=e.size()-1;
25 }
26 int augment(int u, int cur_flow){
27     if(u==T || !cur_flow) return ans+=piS*
28         cur_flow, cur_flow;
29     vis[u]=1;
30     int r=cur_flow, d;
31     for(int i=g[u]; ~i; i=e[i].pre){
32         if(e[i].cap&&!e[i].cost&&vis[e[i].v]){
33             d=augment(e[i].v, std::min(r, e[i].cap))
34                 ;
35             e[i].cap-=d;
36             e[i^1].cap+=d;
37             if(!r==d) break;
38         }
39     }
40     return cur_flow-r;
41 }
```

```

36 }
37 inline bool modlabel(){
38     for(int i=0;i<n;++i)dis[i]=INF;
39     dis[T]=0;
40     static std::deque<int>q;
41     q.push_back(T);
42     while(q.size()){
43         int u=q.front();
44         q.pop_front();
45         int dt;
46         for(int i=g[u];~i;i=e[i].pre){
47             if(e[i^1].cap&&(dt=dis[u]-e[i].cost)<
48                 dis[e[i].v]){
49                 if((dis[e[i].v]=dt)<=dis[q.size()>q.
50                     front():S){}
51                 q.push_front(e[i].v);
52             }else q.push_back(e[i].v);
53         }
54     }
55     for(int u=0;u<n;++u){
56         for(int i=g[u];~i;i=e[i].pre){
57             e[i].cost+=dis[e[i].v]-dis[u];
58         }
59     }
60     piS+=dis[S];
61     return dis[S]<INF;
62 }
63 inline int mincost(){
64     piS=ans=0;
65     while(modlabel()){
66         do memset(vis,0,sizeof(bool)*n);
67         while(augment(S,INF));
68     }
69     return ans;

```

5 Graph

5.1 Arborescence_EV.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct node {
5     int from, to, cost;
6     node(int from=0,int to=0,int cost=0):
7         from(from),to(to),cost(cost){};
8 } edge[M];
9
10 int m, n, m, c;
11 int far[N], In[N], ID[N], vis[N];
12
13 bool MST(int cost,int n,int root)
14 {
15     long long int ans=0;
16     while(true)
17     {
18         for(int i=0;i<n;++i) IN[i].first =
19             INF;
20         for(int i=0;i<m;++i)

```

```

19         if(edge[i].from!=edge[i].to)
20             IN[edge[i].to] = min(IN[edge
21                 [i].to],make_pair(edge[i
22                 ].cost,edge[i].from));
23         for(int i=0;i<n;++i)
24             if(i!=root && IN[i].first==INF)
25                 return false; // NO
26                 Arborescence
27
28         int cntnode = 0;
29         memset(ID,-1,sizeof(ID));
30         memset(vis,-1,sizeof(vis));
31         In[root] = 0;
32         for(int i=0;i<n;++i) ans += IN[i].
33             first;
34         for(int i=0;i<n;++i) {
35             int x;
36             for(x=i;vis[x]!=i&&ID[x]==-1&&x
37                 !=root;x=IN[x].second)
38                 vis[x] = i;
39             if(ID[x]==-1 && x!=root) {
40                 for(int i=IN[x].second;u!=x;
41                     u=IN[u].second)
42                     ID[u] = cntnode;
43                 ++cntnode;
44             }
45         }
46         if(cntnode==0) break; // END
47
48         for(int i=0;i<n;++i)
49             if(ID[i]==-1)
50                 ID[i] = cntnode++;
51
52         for(int i=0;i<m;++i) {
53             int v = edge[i].to;
54             edge[i].from = ID[edge[i].from];
55             edge[i].to = ID[edge[i].to];
56             if(edge[i].from!=edge[i].to)
57                 edge[i].cost -= IN[edge[i].
58                     to].first;
59         }
60         n=cntnode;
61         root=ID[root];
62     }
63     return ans<=cost;
64 }

```

5.2 Augmenting_Path.cpp

```

1 #define MAXN1 505
2 #define MAXN2 505
3 int n1,n2; //n1個點連向n2個點*/
4 int match[MAXN2]; //每個屬於n2的點匹配了哪個
5 點*/
6 vector<int> g[MAXN1]; //圖*/
7 bool vis[MAXN2]; //是否走訪過*/
8 bool dfs(int u){
9     for(size_t i=0;i<g[u].size();++i){
10         int v=g[u][i];
11         if(vis[v])continue;
12         vis[v]=1;
13         if(match[v]==-1||dfs(match[v])){

```

```

13         match[v]=u;
14         return 1;
15     }
16 }
17 return 0;
18 }
19 inline int max_match(){
20     int ans=0;
21     memset(match,-1,sizeof(int)*n2);
22     for(int i=0;i<n1;++i){
23         memset(vis,0,sizeof(bool)*n2);
24         if(dfs(i))++ans;
25     }
26     return ans;
27 }

```

5.3 Augmenting_Path_multiple

```

1 #define MAXN1 1005
2 #define MAXN2 505
3 int n1,n2; //n1個點連向n2個點，其中n2個點可以
4 匹配很多邊
5 vector<int> g[MAXN1]; //圖
6 int c[MAXN2]; //每個屬於n2點最多可以接受幾條
7 匹配邊
8 vector<int> match_list[MAXN2]; //每個屬於n2的
9 點匹配了那些點
10
11 bool vis[MAXN2]; //是否走訪過
12 bool dfs(int u){
13     for(size_t i=0;i<g[u].size();++i){
14         int v=g[u][i];
15         if(vis[v])continue;
16         vis[v]=true;
17         if((int)match_list[v].size()<c[v]){
18             match_list[v].push_back(u);
19             return true;
20         }else{
21             for(size_t j=0;j<match_list[v].size()
22                 ;++j){
23                 int next_u=match_list[v][j];
24                 if(dfs(next_u)){
25                     match_list[v][j]=u;
26                     return true;
27                 }
28             }
29         }
30     }
31     return false;
32 }
33 inline int max_match(){
34     for(int i=0;i<n2;++i)match_list[i].clear()
35     ;
36     int cnt=0;
37     for(int u=0;u<n1;++u){
38         memset(vis,0,sizeof(bool)*n2);
39         if(dfs(u))++cnt;
40     }
41     return cnt;
42 }

```

5.4 blossom_matching.cpp

```

1 #define MAXN 505
2 vector<int>g[MAXN];
3 int pa[MAXN],match[MAXN],st[MAXN],S[MAXN],v[
4     MAXN];
5 int t,n;
6 inline int lca(int x,int y){
7     for(++t;swap(x,y)){
8         if(x==0)continue;
9         if(v[x]==t)return x;
10         v[x]=t;
11         x=st[pa[match[x]]];
12     }
13 }
14 #define qpush(x) q.push(x),S[x]=0
15 inline void flower(int x,int y,int l,queue<
16     int> &q){
17     while(st[x]!=1){
18         pa[x]=y;
19         if(S[y==match[x]]==1)qpush(y);
20         st[x]=st[y]=1,x=pa[y];
21     }
22 }
23 inline bool bfs(int x){
24     for(int i=1;i<n;++i)st[i]=i;
25     memset(S+1,-1,sizeof(int)*n);
26     queue<int>q;qqpush(x);
27     while(q.size()){
28         x=q.front(),q.pop();
29         for(size_t i=0;i<g[x].size();++i){
30             int y=g[x][i];
31             if(S[y]==-1){
32                 pa[y]=x,S[y]=1;
33                 if(!match[y]){
34                     for(int lst;x;y=lst,x=pa[y])
35                         lst=match[x],match[x]=y,match[y]
36                             =x;
37                     return 1;
38                 }
39                 qqpush(match[y]);
40             }else if(!S[y]&&st[y]!=st[x]){
41                 int l=lca(y,x);
42                 flower(y,x,l,q),flower(x,y,l,q);
43             }
44         }
45     }
46     return 0;
47 }
48 inline int blossom(){
49     int ans=0;
50     for(int i=1;i<n;++i)
51         if(!match[i]&&bfs(i))++ans;
52     return ans;
53 }

```

5.5 formula.tex

對於連通圖 G 最大獨立點集的大小設為 $I(G)$ 最大匹配大小設為 $M(G)$ 最小點覆蓋設為 $C_v(G)$ 最小邊覆蓋設為 $C_e(G)$ 對於任意連通圖 $I(G)+C_v(G)=|V|$ $M(G)+C_e(G)=|V|$ 對於連通二分圖 $I(G)=C_v(G)$ $M(G)=C_e(G)$ 最大團 = 補圖的最大獨立集

5.6 graphISO.cpp

```

1 const int MAXN=1005,K=30; //K要夠大
2 const long long A=3,B=11,C=2,D=19,P=0
3 xdefaced;
4 long long f[K+1][MAXN];
5 vector<int> g[MAXN],rg[MAXN];
6 int n;
7 inline void init(){
8     for(int i=0;i<n;++i){
9         f[0][i]=1;
10        g[i].clear();
11        rg[i].clear();
12    }
13    inline void add_edge(int u,int v){
14        g[u].push_back(v);
15        rg[v].push_back(u);
16    }
17    inline long long point_hash(int u){ //O(N)
18        for(int t=1;t<=K;++t){
19            for(int i=0;i<n;++i){
20                f[t][i]=f[t-1][i]*A%P;
21                for(int j:g[i])f[t][i]=(f[t][i]+f[t-1][j]*B%P)%P;
22                for(int j:rg[i])f[t][i]=(f[t][i]+f[t-1][j]*C%P)%P;
23                if(i==u)f[t][i]+=D; //如果圖太大的話，
24                //把這行刪掉，執行一次後f[K]就會是所有點的答案
25                f[t][i]%=P;
26            }
27        }
28        return f[K][u];
29    }
30    inline vector<long long> graph_hash(){
31        vector<long long> ans;
32        for(int i=0;i<n;++i)ans.push_back(
33            point_hash(i)); //O(N^2)
34        sort(ans.begin(),ans.end());
35        return ans;
36    }

```

5.7 KM.cpp

```

1 #define MAXN 100
2 int n;
3 int g[MAXN][MAXN],lx[MAXN],ly[MAXN],slack_y[
4     MAXN];
5 int match_y[MAXN];
6 bool vx[MAXN],vy[MAXN]; //要保證g是完全二分圖
7 bool dfs(int x,bool adjust=1){ //DFS找增廣
8     //路，is=1表示要交換邊
9     if(vx[x])return 0;
10    vx[x]=1;
11    for(int y=0;y<n;++y){
12        if(vy[y])continue;
13        int t=lx[x]+ly[y]-g[x][y];
14        if(t==0){
15            vy[y]=1;

```

```

14        if(match_y[y]==-1||dfs(match_y[y],
15            adjust)){
16            if(adjust)match_y[y]=x;
17            return 1;
18        }else if(slack_y[y]>t)slack_y[y]=t;
19    }
20    return 0;
21 }
22 inline int km(){
23     memset(lx,0,sizeof(int)*n);
24     memset(match_y,-1,sizeof(int)*n);
25     for(int x=0;x<n;++x){
26         lx[x]=0;
27         for(int y=0;y<n;++y){
28             lx[x]=max(lx[x],g[x][y]);
29         }
30     }
31     for(int x=0;x<n;++x){
32         for(int y=0;y<n;++y)slack_y[y]=INT_MAX;
33         memset(vx,0,sizeof(bool)*n);
34         memset(vy,0,sizeof(bool)*n);
35         if(dfs(x))continue;
36         bool flag=1;
37         while(flag){
38             int cut=INT_MAX;
39             for(int y=0;y<n;++y){
40                 if(!vy[y]&&cut>slack_y[y])cut=
41                     slack_y[y];
42             }
43             for(int j=0;j<n;++j){
44                 if(vx[j])lx[j]-=cut;
45                 if(vy[j])ly[j]+=cut;
46                 else slack_y[j]-=cut;
47             }
48             for(int y=0;y<n;++y){
49                 if(!vy[y]&&slack_y[y]==0){
50                     vy[y]=1;
51                     if(match_y[y]==-1||dfs(match_y[y],
52                         0)){
53                         flag=0; //測試成功，有增廣路
54                         break;
55                     }
56                 }
57             }
58             memset(vx,0,sizeof(bool)*n);
59             memset(vy,0,sizeof(bool)*n);
60             dfs(x); //最後要記得將邊翻反轉
61         }
62         int ans=0;
63         for(int y=0;y<n;++y)ans+=g[match_y[y]][y];
64         return ans;
65     }

```

5.8 MaximumClique.cpp

```

1 struct MaxClique{
2     static const int MAXN=105;
3     int N,ans;
4     int g[MAXN][MAXN],dp[MAXN],stk[MAXN][MAXN];
5 }

```

```

5 int sol[MAXN],tmp[MAXN]; //sol[0~ans-1]為答案
6 void init(int n){
7     N=n; //0-base
8     memset(g,0,sizeof(g));
9 }
10 void add_edge(int u,int v){
11     g[u][v]=g[v][u]=1;
12 }
13 int dfs(int ns,int dep){
14     if(!ns){
15         if(dep>ans){
16             ans=dep;
17             memcpy(sol,tmp,sizeof tmp);
18             return 1;
19         }else return 0;
20     }
21     for(int i=0;i<ns;++i){
22         if(dep+ns-i==ans)return 0;
23         int u=stk[dep][i],cnt=0;
24         if(dep+dp[u]<=ans)return 0;
25         for(int j=i+1;j<ns;++j){
26             int v=stk[dep][j];
27             if(g[u][v])stk[dep+1][cnt++]=v;
28         }
29         tmp[dep]=u;
30         if(dfs(cnt,dep+1))return 1;
31     }
32     return 0;
33 }
34 int clique(){
35     int u,v,ns;
36     for(ans=0,u=N-1;u>=0;--u){
37         for(ns=0,tmp[0]=u,v=u+1;v<N;++v)
38             if(g[u][v])stk[1][ns++]=v;
39         dfs(ns,1),dp[u]=ans;
40     }
41     return ans;
42 }
43 };

```

5.9 Minimum_General_Weighted

```

1 struct Graph {
2     // Minimum General Weighted Matching (
3     // Perfect Match) 0-base
4     static const int MXN = 105;
5
6     int n, edge[MXN][MXN];
7     int match[MXN],dis[MXN],onstk[MXN];
8     vector<int> stk;
9
10    void init(int _n) {
11        n = _n;
12        for (int i=0; i<n; i++)
13            for (int j=0; j<n; j++)
14                edge[i][j] = 0;
15
16    void add_edge(int u, int v, int w) {
17        edge[u][v] = edge[v][u] = w;
18    }
19
20    bool SPFA(int u){
21        if (onstk[u]) return true;

```

```

20    stk.push_back(u);
21    onstk[u] = 1;
22    for (int v=0; v<n; v++){
23        if (u != v && match[u] != v && !onstk[
24            v]){
25            int m = match[v];
26            if (dis[m] > dis[u] - edge[v][m] +
27                edge[u][v]){
28                dis[m] = dis[u] - edge[v][m] +
29                    edge[u][v];
30                onstk[v] = 1;
31                stk.push_back(v);
32                if (SPFA(m)) return true;
33                stk.pop_back();
34                onstk[v] = 0;
35            }
36        }
37        onstk[u] = 0;
38        stk.pop_back();
39        return false;
40    }
41    int solve() {
42        // find a match
43        for (int i=0; i<n; i+=2){
44            match[i] = i+1;
45            match[i+1] = i;
46        }
47        for(;;){
48            int found = 0;
49            for (int i=0; i<n; i++){
50                dis[i] = onstk[i] = 0;
51                for (int i=0; i<n; i++){
52                    stk.clear();
53                    if (!onstk[i] && SPFA(i)){
54                        found = 1;
55                        while (stk.size())>=2){
56                            int u = stk.back(); stk.pop_back
57                                ();
58                            int v = stk.back(); stk.pop_back
59                                ();
60                            match[u] = v;
61                            match[v] = u;
62                        }
63                    }
64                }
65                if (!found) break;
66            }
67            int ret = 0;
68            for (int i=0; i<n; i++){
69                ret += edge[i][match[i]];
70            }
71            ret /= 2;
72            return ret;
73        }
74    }graph;

```

5.10 Rectilinear_Steiner_tree.c

```

1 //平面曼哈頓最小生成樹構造圖(去除非必要邊)
2 #include<vector>
3 #include<algorithm>
4 #define T int

```

```

5 #define INF 0x3f3f3f3f
6 struct point{
7     T x,y;
8     int id;//每個點的編號都要不一樣，從0開始編號
9     point(){}
10    T dist(const point &p)const{
11        return std::abs(x-p.x)+std::abs(y-p.y);
12    }
13 };
14 inline bool cmpx(const point &a,const point &b){
15     return a.x<b.x||(a.x==b.x&&a.y<b.y);
16 }
17 struct edge{
18     int u,v;
19     T cost;
20     edge(int u,int v,const T&c):u(u),v(v),cost(c){}
21     bool operator<(const edge&e)const{
22         return cost<e.cost;
23     }
24 };
25 struct bit_node{
26     T mi;
27     int id;
28     bit_node(const T&mi=INF,int id=-1):mi(mi),id(id){}
29 };
30 std::vector<bit_node> bit;
31 inline void bit_update(int i,const T&data, int id){
32     for(;;i=i&(-i)){
33         if(data<bit[i].mi)bit[i]=bit_node(data,id);
34     }
35 }
36 inline int bit_find(int i,int m){
37     bit_node x;
38     for(;;i=i&(-i)){
39         if(bit[i].mi<x.mi)x=bit[i];
40     }
41     return x.id;
42 }
43 inline std::vector<edge> build_graph(int n, point p[]){
44     std::vector<edge> e;//回傳的邊就可以用來求最小生成樹
45     for(int dir=0;dir<4;dir++){//4種座標交換
46         if(dir%2){
47             for(int i=0;i<n;i++)std::swap(p[i].x,p[i].y);
48         }else if(dir==2){
49             for(int i=0;i<n;i++)p[i].x=-p[i].x;
50         }
51         std::sort(p,p+n,cmpx);
52         std::vector<T>ga(n),gb;
53         for(int i=0;i<n;i++)ga[i]=p[i].y-p[i].x;
54         gb=ga;
55         std::sort(gb.begin(),gb.end());
56         gb.resize(std::unique(gb.begin(),gb.end())-gb.begin());
57         int m=gb.size();
58         bit=std::vector<bit_node>(m+1);
59         for(int i=n-1;i>0;--i){

```

```

60         int pos=std::lower_bound(gb.begin(),gb.end(),ga[i])-gb.begin()+1;
61         int ans=bit_find(pos,m);
62         if(~ans)e.push_back(edge(p[i].id,p[ans].id,p[i].dist(p[ans])));
63         bit_update(pos,p[i].x+p[i].y,i);
64     }
65     return e;
66 }
67 }

```

5.11 treeISO.cpp

```

1 const int MAXN=100005;
2 const long long X=12327,P=0xdefaced;
3 vector<int> g[MAXN];
4 bool vis[MAXN];
5 long long dfs(int u){
6     vis[u]=1;
7     vector<long long> tmp;
8     for(auto v:g[u])if(!vis[v])tmp.push_back(dfs(v));
9     if(tmp.empty())return 177;
10    long long ret=4931;
11    sort(tmp.begin(),tmp.end());
12    for(auto v:tmp)ret=((ret*X)^v)%P;
13    return ret;
14 }

```

5.12 一般圖最大權匹配.cpp

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define INF INT_MAX
4 #define MAXN 400
5 struct edge{
6     int u,v,w;
7     edge(){}
8     edge(int u,int v,int w):u(u),v(v),w(w){}
9 };
10 int n,n_x;
11 edge g[MAXN*2+1][MAXN*2+1];
12 int lab[MAXN*2+1];
13 int match[MAXN*2+1],slack[MAXN*2+1],st[MAXN*2+1],pa[MAXN*2+1];
14 int flower_from[MAXN*2+1][MAXN+1],S[MAXN*2+1],vis[MAXN*2+1];
15 vector<int> flower[MAXN*2+1];
16 queue<int> q;
17 inline int e_delta(const edge &e){ // does not work inside blossoms
18     return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
19 }
20 inline void update_slack(int u,int x){
21     if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]][x]))slack[x]=u;
22 }
23 inline void set_slack(int x){
24     slack[x]=0;
25     for(int u=1;u<n;u++){

```

```

26     if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)update_slack(u,x);
27 }
28 void q_push(int x){
29     if(x<n)q.push(x);
30     else for(size_t i=0;i<flower[x].size();i++)q_push(flower[x][i]);
31 }
32 inline void set_st(int x,int b){
33     st[x]=b;
34     if(x>n)for(size_t i=0;i<flower[x].size();i++)set_st(flower[x][i],b);
35 }
36 inline int get_pr(int b,int xr){
37     int pr=find(flower[b].begin(),flower[b].end(),xr)-flower[b].begin();
38     if(pr%2==1){//檢查他在前一層圖是奇點還是偶點
39         reverse(flower[b].begin()+1,flower[b].end());
40         return (int)flower[b].size()-pr;
41     }else return pr;
42 }
43 inline void set_match(int u,int v){
44     match[u]=g[u][v].v;
45     if(u>n){
46         edge e=g[u][v];
47         int xr=flower_from[u][e.u],pr=get_pr(u,xr);
48         for(int i=0;i<pr;i++)set_match(flower[u][i],flower[u][i^1]);
49         set_match(xr,v);
50         rotate(flower[u].begin(),flower[u].begin()+pr,flower[u].end());
51     }
52 }
53 inline void augment(int u,int v){
54     for(;;){
55         int xnv=st[match[u]];
56         set_match(u,v);
57         if(!xnv)return;
58         set_match(xnv,st[pa[xnv]]);
59         u=st[pa[xnv]],v=xnv;
60     }
61 }
62 inline int get_lca(int u,int v){
63     static int t=0;
64     for(++t;u||v;swap(u,v)){
65         if(u==0)continue;
66         if(vis[u]==t)return u;
67         vis[u]=t;//這種方法可以不用清空v陣列
68         u=st[match[u]];
69         if(u)u=st[pa[u]];
70     }
71     return 0;
72 }
73 inline void add_blossom(int u,int lca,int v){
74     {
75         int b=n+1;
76         while(b<=n_x&&st[b]==b)b++;
77         if(b>n_x)b=n_x;
78         lab[b]=0,S[b]=0;
79         match[b]=match[lca];
80         flower[b].clear();

```

```

81         flower[b].push_back(lca);
82         for(int x=u,y;lca;x=st[pa[y]])
83             flower[b].push_back(x),flower[b].push_back(y=st[match[x]]),q_push(y);
84         reverse(flower[b].begin()+1,flower[b].end());
85         for(int x=v,y;lca;x=st[pa[y]])
86             flower[b].push_back(x),flower[b].push_back(y=st[match[x]]),q_push(y);
87         set_st(b,b);
88         for(int x=1;x<=n_x;x++)g[b][x].w=g[x][b].w=0;
89         for(int x=1;x<=n;u++)flower_from[b][x]=0;
90         for(size_t i=0;i<flower[b].size();i++){
91             int xs=flower[b][i];
92             for(int x=1;x<=n_x;x++){
93                 if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b][x]))
94                     g[b][x]=g[xs][x],g[x][b]=g[x][xs];
95             }
96             if(flower_from[xs][x])flower_from[b][x]=xs;
97         }
98         set_slack(b);
99     }
100     inline void expand_blossom(int b){ // S[b] == 1
101         for(size_t i=0;i<flower[b].size();i++){
102             set_st(flower[b][i],flower[b][i]);
103             int xr=flower_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
104             for(int i=0;i<pr;i+=2){
105                 int xs=flower[b][i],xns=flower[b][i+1];
106                 pa[xs]=g[xns][xs].u;
107                 S[xs]=1,S[xns]=0;
108                 slack[xs]=0,set_slack(xns);
109                 q_push(xns);
110             }
111             S[xr]=1,pa[xr]=pa[b];
112             for(size_t i=pr+1;i<flower[b].size();i++){
113                 int xs=flower[b][i];
114                 S[xs]=-1,set_slack(xs);
115             }
116             st[b]=0;
117         }
118         inline bool on_found_edge(const edge &e){
119             int u=st[e.u],v=st[e.v];
120             if(S[v]==-1){
121                 pa[v]=e.u,S[v]=1;
122                 int nu=st[match[v]];
123                 slack[v]=slack[nu]=0;
124                 S[nu]=0,q_push(nu);
125             }else if(S[v]==0){
126                 int lca=get_lca(u,v);
127                 if(!lca){
128                     augment(u,v),augment(v,u);
129                     return true;
130                 }else add_blossom(u,lca,v);
131             }
132             return false;
133         }
134         inline bool matching(){
135             memset(S+1,-1,sizeof(int)*n_x);
136             memset(slack+1,0,sizeof(int)*n_x);
137             q=queue<int>();
138             for(int x=1;x<=n_x;x++){

```

```

139 if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,
    q_push(x);
140 if(q.empty())return false;
141 for(;;){
142     while(q.size()){
143         int u=q.front();q.pop();
144         if(S[st[u]]==1)continue;
145         for(int v=1;v<=n;v++){
146             if(g[u][v].w>0&&st[u]!=st[v]){
147                 if(e_delta(g[u][v])==0){
148                     if(on_found_edge(g[u][v]))return
149                         true;
150                 }else update_slack(u,st[v]);
151             }
152             int d=INF;
153             for(int b=n+1;b<=n_x;v++){
154                 if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
155             }
156             for(int x=1;x<=n_x;v++){
157                 if(st[x]==x&&slack[x]){
158                     if(S[x]==-1)d=min(d,e_delta(g[slack[x]]
159                         [x]));
160                     else if(S[x]==0)d=min(d,e_delta(g[
161                         slack[x]][x])/2);
162                 }
163             }
164             for(int u=1;u<=n;v++){
165                 if(S[st[u]]==0){
166                     if(lab[u]<=d)return 0;
167                     lab[u]=d;
168                 }else if(S[st[u]]==1)lab[u]+=d;
169             }
170             for(int b=n+1;b<=n_x;v++){
171                 if(st[b]==b){
172                     if(S[st[b]]==0)lab[b]+=d*2;
173                     else if(S[st[b]]==1)lab[b]-=d*2;
174                 }
175             }
176             q=queue<int>();
177             for(int x=1;x<=n_x;v++){
178                 if(st[x]==x&&slack[x]&&st[slack[x]]!=x
179                     &&e_delta(g[slack[x]][x])==0)
180                 if(on_found_edge(g[slack[x]][x]))
181                     return true;
182             }
183             for(int b=n+1;b<=n_x;v++){
184                 if(st[b]==b&&S[b]==1&&lab[b]==0)
185                     expand_blossom(b);
186             }
187             return false;
188         }
189     }
190 }
191 inline pair<long long,int> weight_blossom(){
192     memset(match+1,0,sizeof(int)*n);
193     n_x=n;
194     int n_matches=0;
195     long long tot_weight=0;
196     for(int u=0;u<=n;v++){
197         st[u]=u,flower[u].
198         clear();
199     }
200     int w_max=0;
201     for(int u=1;u<=n;v++){
202         for(int v=1;v<=n;v++){
203             flower_from[u][v]=(u==v?0:0);
204             w_max=max(w_max,g[u][v].w);
205         }
206     }
207     for(int u=1;u<=n;v++){
208         lab[u]=w_max;
209         while(matching())n_matches++;
210     }
211     for(int u=1;u<=n;v++){
212         if(match[u]&&match[u]<u)

```

```

196     tot_weight+=g[u][match[u]].w;
197     return make_pair(tot_weight,n_matches);
198 }
199 inline void init_weight_graph(){
200     for(int u=1;u<=n;v++){
201         for(int v=1;v<=n;v++){
202             g[u][v]=edge(u,v,0);
203         }
204     }
205     int main(){
206         int m;
207         scanf("%d%d",&n,&m);
208         init_weight_graph();
209         for(int i=0;i<=m;v++){
210             int u,v,w;
211             scanf("%d%d%d",&u,&v,&w);
212             g[u][v].w=g[v][u].w=w;
213         }
214         printf("%Lld\n",weight_blossom().first);
215         for(int u=1;u<=n;v++){
216             puts("");
217         }
218         return 0;
219     }

```

6 language

6.1 CNF.cpp

```

1 #define MAXN 55
2 struct CNF{
3     int s,x,y;//s->xy | s->x, if y==1
4     int cost;
5     CNF(){}
6     CNF(int s,int x,int y,int c):s(s),x(x),y(y)
7         ,cost(c){}
8 };
9 int state;//規則數量
10 map<char,int> rule;//每個字元對應到的規則
11 小寫字母為終端字元
12 vector<CNF> cnf;
13 inline void init(){
14     state=0;
15     rule.clear();
16     cnf.clear();
17 }
18 inline void add_to_cnf(char s,const string &
19     p,int cost){
20     //加入一個s -> <p>的文法，代價為cost
21     if(rule.find(s)==rule.end())rule[s]=state++;
22     for(auto c:p)if(rule.find(c)==rule.end())
23         rule[c]=state++;
24     if(p.size()==1){
25         cnf.push_back(CNF(rule[s],rule[p[0]],-1,
26             cost));
27     }else{
28         int left=rule[s];
29         int sz=p.size();
30         for(int i=0;i<sz-2;v++){
31             cnf.push_back(CNF(left,rule[p[i]],
32                 state,0));
33         }
34     }
35 }

```

```

27     left=state++;
28 }
29 cnf.push_back(CNF(left,rule[p[sz-2]],
30     rule[p[sz-1]],cost));
31 }
32 vector<long long> dp[MAXN][MAXN];
33 vector<bool> neg_INF[MAXN][MAXN];//如果花費
34 是負的可能會有無限小的情形
35 inline void relax(int l,int r,const CNF &c,
36     long long cost,bool neg_c=0){
37     if(!neg_INF[l][r][c.s]&&(neg_INF[l][r][c.x]
38         ||cost<dp[l][r][c.s])){
39         if(neg_c||neg_INF[l][r][c.x]){
40             dp[l][r][c.s]=0;
41             neg_INF[l][r][c.s]=true;
42         }else dp[l][r][c.s]=cost;
43     }
44     inline void bellman(int l,int r,int n){
45         for(int k=1;k<=state;v++){
46             for(auto c:cnf)
47                 if(c.y==1)relax(l,r,c,dp[l][r][c.x]+c
48                     .cost,k==n);
49         }
50     }
51     inline void cyk(const vector<int> &tok){
52         for(int i=0;i<(int)tok.size();v++){
53             for(int j=0;j<(int)tok.size();v++){
54                 dp[i][j]=vector<long long>(state+1,
55                     INT_MAX);
56                 neg_INF[i][j]=vector<bool>(state+1,
57                     false);
58             }
59             dp[i][i][tok[i]]=0;
60             bellman(i,i,tok.size());
61         }
62     }
63     for(int r=1;r<(int)tok.size();v++){
64         for(int l=r-1;l>=0;v++){
65             for(int k=1;k<=r;v++){
66                 for(auto c:cnf)
67                     if(~c.y)relax(l,r,c,dp[l][k][c.x]+
68                         dp[k+1][r][c.y]+c.cost);
69                 bellman(l,r,tok.size());
70             }
71         }
72     }
73 }

```

6.2 earley.cpp

```

1 struct Rule{
2     vector<vector<Rule*> > p;
3     void add(const vector<Rule*> &l){
4         p.push_back(l);
5     }
6 };
7 map<string,Rule*> NameRule;
8 map<Rule*,string> RuleName;
9 inline void init_Rule(){
10     for(auto r:RuleName)delete r.first;
11     RuleName.clear();
12     NameRule.clear();
13 }

```

```

14 inline Rule *add_rule(const string &s){
15     if(NameRule.find(s)!=NameRule.end())return
16         NameRule[s];
17     Rule *r=new Rule();
18     RuleName[r]=s;
19     NameRule[s]=r;
20     return r;
21 }
22 typedef vector<Rule*> production;
23 struct State{
24     Rule *r;
25     int rid,dot_id,start,end;
26     State(Rule *r,int rid,int dot,int start):r(
27         r),rid(rid),dot_id(dot),start(start),
28         end(-1){}
29     State(Rule *r=0,int col=0):r(r),rid(-1),
30         dot_id(-1),start(-1),end(col){}
31     bool completed()const{
32         return rid==-1||dot_id==(int)r->p[rid].
33             size();
34     }
35     Rule *next_term()const{
36         if(completed())return 0;
37         return r->p[rid][dot_id];
38     }
39 }
40 bool operator<(const State& b)const{
41     if(start!=b.start)return start<b.start;
42     if(dot_id!=b.dot_id)return dot_id<b.
43         dot_id;
44     if(r!=b.r)return r<b.r;
45     return rid<b.rid;
46 }
47 void print()const{
48     cout<<RuleName[r]<<"-";
49     if(rid!=-1)for(size_t i=0;v++){
50         if((int)i==dot_id)cout<<" "<<"$";
51         if(i==r->p[rid].size())break;
52         cout<<" "<<RuleName[r->p[rid][i]];
53     }
54     cout<<" "<<"["<<start<<","<<end<<"]<<
55         endl;
56 }
57 }
58 struct Column{
59     Rule *term;
60     string value;
61     vector<State> s;
62     map<State,set<pair<State,State> > > div;
63     //div比較像一棵 左右兄子的樹
64     Column(Rule *r,const string &s):term(r),
65         value(s){}
66     Column(){}
67     bool add(const State &st,int col){
68         if(div.find(st)==div.end()){
69             div[st];
70             s.push_back(st);
71             s.back().end=col;
72             return true;
73         }else return false;
74     }
75 }
76 inline vector<Column> lexer(string text){
77     //tokenize，要自己寫，以下為範例
78     //他會把 input stream 變成 token stream
79     就是 (terminal,value)pair

```

```

70 vector<Column> token;
71 replace(text.begin(),text.end(),',',' ');
72 stringstream ss(text);
73 while(ss>>text){
74     if(text=="a"||text=="of")continue;
75     if(text=="List"){
76         token.push_back(Column(NameRule["(",",",
77             ")"]);
78     }else if(text=="and"){
79         token.push_back(Column(NameRule[" "],",",
80             ")"));
81     }else token.push_back(Column(NameRule["T",
82         " "],text));
83 }
84 return token;
85 }
86 vector<Column> table;
87 inline void predict(int col,Rule *rul){
88     for(size_t i=0;i<rul->p.size();++i){
89         table[col].add(State(rul,i,0,col),col);
90     }
91 }
92 inline void scan(int col,const State &s,Rule
93     *r){
94     if(r!=table[col].term)return;
95     State ns(s.r,s.rid,s.dot_id+1,s.start);
96     table[col].add(ns,col);
97     table[col].div[ns].insert(make_pair(s,
98         State(r,col)));
99 }
100 inline void complete(int col,const State &s)
101 {
102     for(size_t i=0;i<table[s.start].s.size()
103         ;++i){
104         State &st=table[s.start].s[i];
105         Rule *term=st.next_term();
106         if(!term||term->p.size()==0)continue;
107         if(term==s.r){
108             State nst(st.r,st.rid,st.dot_id+1,st.
109                 start);
110             table[col].add(nst,col);
111             table[col].div[nst].insert(make_pair(
112                 st,s));
113         }
114     }
115 }
116 inline pair<bool,State> parse(Rule *GAMMA,
117     const vector<Column> &token){
118     table.resize(token.size()+1);
119     for(size_t i=0;i<token.size();++i)table[i
120         +1]=Column(token[i]);
121     table[0]=Column();
122     table[0].add(State(GAMMA,0,0,0),0);
123     for(size_t i=0;i<table.size();++i){
124         for(size_t j=0;j<table[i].s.size();++j){
125             State state=table[i].s[j];
126             if(state.completed())complete(i,state)
127             ;
128         }
129         Rule *term=state.next_term();
130         if(term->p.size())predict(i,term);
131         else if(i+1<table.size())scan(i+1,
132             state,term);
133     }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

7 Number_Theory

7.1 basic.cpp

```

1 typedef long long int LL;
2 template<typename T>
3 void gcd(const T &a,const T &b,T &d,T &x,T &y){
4     if(!b) d=a,x=1,y=0;
5     else gcd(b,a%b,d,y,x), y-=x*(a/b);
6 }
7
8 const int MAXPRIME = 1000000;
9 int iscom[MAXPRIME], prime[MAXPRIME],
10 primecnt;
11 int phi[MAXPRIME], mu[MAXPRIME];
12 void sieve(void)
13 {
14     memset(iscom,0,sizeof(iscom));
15     primecnt = 0;
16     phi[1] = mu[1] = 1;
17     for(int i=2;i<MAXPRIME;++i) {
18         if(!iscom[i]) {
19             prime[primecnt++] = i;
20             mu[i] = -1;
21             phi[i] = i-1;
22         }
23         for(int j=0;j<primecnt;++j) {
24             int k = i * prime[j];
25             if(k>MAXPRIME) break;
26             iscom[k] = prime[j];
27             if(i%prime[j]==0) {
28                 mu[k] = 0;
29                 phi[k] = phi[i] * prime[j];
30                 break;
31             } else {
32                 mu[k] = -mu[i];
33                 phi[k] = phi[i] * (prime[j]
34                     -1);
35             }
36         }
37     }
38 }
39
40 bool g_test(const LL &g, const LL &p, const
41     vector<LL> &v) {
42     for(int i=0;i<v.size();++i)
43         if(modexp(g,(p-1)/v[i],p)==1)
44             return false;
45     return true;
46 }
47
48 LL primitive_root(const LL &p) {
49     if(p==2) return 1;
50     vector<LL> v;
51     Factor(p-1,v);
52     v.erase(unique(v.begin(), v.end()), v.
53         end());
54     for(LL g=2;g<p;++g)
55         if(g_test(g,p,v))
56             return g;
57     puts("primitive_root NOT FOUND");
58     return -1;
59 }
60
61 int Legendre(const LL &a, const LL &p) {
62     return modexp(a%p,(p-1)/2,p);
63 }
64
65 LL inv(const LL &a, const LL &n) {
66     LL d,x,y;
67     gcd(a,n,d,x,y);
68     return d==1 ? (x+n)%n : -1;
69 }
70
71 LL log_mod(const LL &a, const LL &b, const
72     LL &p) {
73     // a ^ x = b ( mod p )
74     int m=sqrt(p+.5), e=1;
75     LL v=inv(modexp(a,m,p), p);
76     map<LL,int> x;
77     x[1]=0;
78     for(int i=1;i<m;++i) {
79         e = LLMul(e,a,p);
80         if(!x.count(e)) x[e] = i;
81     }
82     for(int i=0;i<m;++i) {
83         if(x.count(b)) return i*m + x[b];
84         b = LLMul(b,v,p);
85     }
86     return -1;
87 }
88
89 LL Tonelli_Shanks(const LL &n, const LL &p)
90 {
91     // x^2 = n ( mod p )
92     if(n==0) return 0;
93     if(Legendre(n,p)!=1) while(1) { puts("
94         SQRRT ROOT does not exist"); }
95     int S = 0;
96     LL Q = p-1;
97     while( !(Q&1) ) { Q>>=1; ++S; }
98     if(S==1) return modexp(n%p,(p+1)/4,p);
99     LL z = 2;
100     for(; Legendre(z,p)!=-1;++z)
101         LL c = modexp(z,Q,p);
102 }

```



```

92 LL R = modexp(n%p, (Q+1)/2, p), t = modexp
    (n%p, Q, p);
93 int M = S;
94 while(1) {
95     if(t==1) return R;
96     LL b = modexp(c, 1L<<(M-i-1), p);
97     R = LLMul(R, b, p);
98     t = LLMul(LLMul(b, b, p), t, p);
99     c = LLMul(b, b, p);
100    M = i;
101 }
102 return -1;
103 }

```

7.2 bit_set.cpp

```

1 void sub_set(int S){
2     int sub=S;
3     do{
4         //對某集合的子集合的處理
5         sub=(sub-1)&S;
6     }while(sub!=S);
7 }
8 void k_sub_set(int k, int n){
9     int comb=(1<<k)-1, S=1<<n;
10    while(comb<S){
11        //對大小為k的子集合的處理
12        int x=comb&-comb, y=comb+x;
13        comb=((comb&-y)/x>>1)|y;
14    }
15 }

```

7.3 cantor_expansion.cpp

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define MAXN 11
4 int factorial[MAXN];
5 inline void init(){
6     factorial[0]=1;
7     for(int i=1; i<=MAXN; ++i) factorial[i]=
        factorial[i-1]*i;
8 }
9 inline int encode(const std::vector<int> &s)
10 {
11     int n=s.size(), res=0;
12     for(int i=0; i<n; ++i){
13         int t=0;
14         for(int j=i+1; j<n; ++j)
15             if(s[j]<s[i]) ++t;
16         res+=t*factorial[n-i-1];
17     }
18     return res;
19 }
20 inline std::vector<int> decode(int a, int n){
21     std::vector<int> res;
22     std::vector<bool> vis(n, 0);
23     for(int i=n-1; i>=0; --i){
24         int t=a/factorial[i], j;

```

```

24     for(j=0; j<n; ++j)
25         if(!vis[j]){
26             if(t==0) break;
27             --t;
28         }
29     res.push_back(j);
30     vis[j]=1;
31     a%=factorial[i];
32 }
33 return res;
34 }
35 int main(){
36     init();
37     vector<int> p={0,1,2,3,4,5,6,7,8};
38     for(int i=0; i<factorial[9]; ++i){
39         vector<int> s=decode(i, 9);
40         if(s!=p) puts("XX");
41         next_permutation(p.begin(), p.end());
42     }
43     return 0;
44 }

```

7.4 Chinese_remainder_theorem

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #ifndef CHINESE_REMAINDER_THEOREM
4 #define CHINESE_REMAINDER_THEOREM
5 template<typename T>
6 inline T Euler(T n){
7     T ans=n;
8     for(T i=2; i*i<=n; ++i){
9         if(n%i==0){
10             ans=ans/i*(i-1);
11             while(n%i==0) n/=i;
12         }
13     }
14     if(n>1) ans=ans*n*(n-1);
15     return ans;
16 }
17 template<typename T>
18 inline T pow_mod(T n, T k, T m){
19     T ans=1;
20     for(n=(n>=m?n%m:n); k>=1){
21         if(k&1) ans=ans*n%m;
22         n=n*n%m;
23     }
24     return ans;
25 }
26 template<typename T>
27 inline T crt(std::vector<T> &m, std::vector<T>
    &a){
28     T M=1, tM, ans=0;
29     for(int i=0; i<(int)m.size(); ++i) M*=m[i];
30     for(int i=0; i<(int)a.size(); ++i){
31         tM=M/m[i];
32         ans=(ans+(a[i]*tM%M)*pow_mod(tM, Euler(m[
33             i])-1, m[i]))%M;
34         /*如果m[i]是質數·Euler(m[i])-1=m[i]-2·
35         就不用算Euler了*/
36     }
37     return ans;
38 }

```

```

36 }
37 #endif
38 int n;
39 vector<long long> a, m;
40 int main(){
41     while(~scanf("%d", &n)){
42         for(int i=0; i<n; ++i){
43             long long x, y;
44             scanf("%lld%lld", &x, &y);
45             m.push_back(x);
46             a.push_back(y);
47         }
48         long long ans=crt(m, a);
49         printf("%lld\n", ans);
50         for(int i=0; i<n; ++i) printf("%lld %lld\n",
            m[i], ans%m[i]);
51         m.clear();
52         a.clear();
53     }
54     return 0;
55 }
56 /*
57 4
58 199 198
59 200 199
60 201 197
61 137 88
62 2
63 265163 465
64 66546165 7122
65 5
66 379 46
67 853 852
68 971 777
69 659 128
70 281 256
71 4
72 6359 1
73 4877 5
74 1627 6
75 8941 7122
76 */

```

7.5 enumerate.cpp

```

1 void all_divdown(const LL &n) { // all n/x
2     for(LL a=1; a<=n; a=n/(n/(a+1))) {
3         // dosomething;
4     }
5 }

```

7.6 eulerphi.cpp

```

1 int eulerPhi(int n){
2     int m = sqrt(n+0.5);
3     int res=n;
4     for(int i=2; i<=m; ++i){
5         if(n%i==0){
6             res = res*(i-1)/i;
7             while(n%i==0) n/=i;

```

```

8         }
9     }
10    if(n>1) res = res*(n-1)/n;
11    return res;
12 }
13 vector<int> phiTable(int n){
14     vector<int> phi(n+1, 0);
15     phi[1] = 1;
16     for(int i=2; i<=n; ++i) if(!phi[i])
17         for(int j=i; j<=n; j+=i){
18             if(!phi[j]) phi[j] = j;
19             phi[j] = phi[j]*(i-1)/i;
20         }
21     return phi;
22 }
23 }

```

7.7 Factor.cpp

```

1 LL LLMul(LL a, LL b, const LL &mod) {
2     LL ans=0;
3     while(b) {
4         if(b&1) {
5             ans+=a;
6             if(ans>=mod) ans-=mod;
7         }
8         a<<=1, b>>=1;
9         if(a>=mod) a-=mod;
10    }
11    return ans;
12 }
13 inline long long mod_mul(long long a, long
    long b, long long m){
14     a%=m, b%=m;
15     long long y=(long long)((double)a*b/m+0.5)
        ;/* fast for m < 2^58 */
16     long long r=(a*b-y*m)%m;
17     return r<0?r+m:r;
18 }
19 template<typename T>
20 inline T pow(T a, T b, T mod){ // a^b%mod
21     T ans=1;
22     for(; b; b=mod_mul(a, a, mod), b>>=1)
23         if(b&1) ans=mod_mul(ans, a, mod);
24     return ans;
25 }
26 int sprp[3]={2, 7, 61}; //int範圍可解
27 int llsp[7]={2, 325, 9375, 28178, 450775, 9780504, 17952650}
    ;//至少unsigned long long範圍
28 template<typename T>
29 inline bool isprime(T n, int *sprp, int num){
30     if(n==2) return 1;
31     if(n<2 || n%2==0) return 0;
32     int t=0;
33     T u=n-1;
34     for(; u%2==0; ++t) u>>=1;
35     for(int i=0; i<num; ++i){
36         T a=sprp[i]%n;
37         if(a==0 || a==1 || a==n-1) continue;
38         T x=pow(a, u, n);
39         if(x==1 || x==n-1) continue;

```

```

40 for(int j=0;j<t;++j){
41     x=mod_mul(x,x,n);
42     if(x==1)return 0;
43     if(x==n-1)break;
44 }
45 if(x==n-1)continue;
46 return 0;
47 }
48 return 1;
49 }
50 LL func(const LL n,const LL mod,const int c)
51 {
52     return (LLmul(n,n,mod)+c+mod)%mod;
53 }
54 LL pollorroho(const LL n, const int c) { //循環
55     環節長度
56     LL a=1, b=1;
57     a=func(a,n,c)%n;
58     b=func(b,n,c)%n; b=func(b,n,c)%n;
59     while(gcd(abs(a-b),n)==1) {
60         a=func(a,n,c)%n;
61         b=func(b,n,c)%n; b=func(b,n,c)%n;
62     }
63     return gcd(abs(a-b),n);
64 }
65 void prefactor(LL &n, vector<LL> &v) {
66     for(int i=0;i<12;++i) {
67         while(n%prime[i]==0) {
68             v.push_back(prime[i]);
69             n/=prime[i];
70         }
71     }
72 }
73 void smallfactor(LL n, vector<LL> &v) {
74     if(n<MAXPRIME) {
75         while(isp[(int)n]) {
76             v.push_back(isp[(int)n]);
77             n/=isp[(int)n];
78         }
79     }
80     v.push_back(n);
81 } else {
82     for(int i=0;i<primecnt&&prime[i]*
83         prime[i]<=n;++i) {
84         while(n%prime[i]==0) {
85             v.push_back(prime[i]);
86             n/=prime[i];
87         }
88     }
89     if(n!=1) v.push_back(n);
90 }
91 }
92 void comfactor(const LL &n, vector<LL> &v) {
93     if(n<1e9) {
94         smallfactor(n,v);
95         return;
96     }
97     if(Isprime(n)) {
98         v.push_back(n);
99         return;
100 }
101 }

```

```

102 LL d;
103 for(int c=3; c<=n; ++c) {
104     d = pollorroho(n,c);
105     if(d!=n) break;
106 }
107 comfactor(d,v);
108 comfactor(n/d,v);
109 }
110 void Factor(const LL &x, vector<LL> &v) {
111     LL n = x;
112     if(n==1) { puts("Factor 1"); return; }
113     prefactor(n,v);
114     if(n==1) return;
115     comfactor(n,v);
116     sort(v.begin(),v.end());
117 }
118 void AllFactor(const LL &n,vector<LL> &v) {
119     vector<LL> tmp;
120     Factor(n,tmp);
121     v.clear();
122     v.push_back(1);
123     int len;
124     LL now=1;
125     for(int i=0;i<tmp.size();++i) {
126         if(i==0 || tmp[i]!=tmp[i-1]) {
127             len = v.size();
128             now = 1;
129         }
130         now*=tmp[i];
131         for(int j=0;j<len;++j)
132             v.push_back(v[j]*now);
133     }
134 }
135 }
136 }

```

7.8 FFT.cpp

```

1 template<typename T,typename VT=std::vector<
2     std::complex<T>>>
3 struct FFT{
4     const T pi;
5     FFT(const T pi=acos((T)-1)):pi(pi){}
6     unsigned int bit_reverse(unsigned int a,
7         int len){
8         a=((a&0x55555555U)<<1)|((a&0xAAAAAAU)
9             >>1);
10        a=((a&0x33333333U)<<2)|((a&0xCCCCCU)
11            >>2);
12        a=((a&0x0F0F0F0FU)<<4)|((a&0xF0F0F0FU)
13            >>4);
14        a=((a&0x00FF00FFU)<<8)|((a&0xFF00FF00U)
15            >>8);
16        a=((a&0x0000FFFFU)<<16)|((a&0xFFFF0000U)
17            >>16);
18        return a>>(32-len);
19    }
20    void fft(bool is_inv,VT &in,VT &out,int N)
21    {
22        int bitlen=std::lg(N),num=is_inv?-1:1;
23        for(int i=0;i<N;++i)out[bit_reverse(i,
24            bitlen)]=in[i];
25        for(int step=2;step<=N;step<=1){

```

```

17 const int mh=step>>1;
18 for(int i=0;i<mh;++i){
19     std::complex<T> wi=exp(std::complex<
20         T>(0,i*num*pi/mh));
21     for(int j=i;j<N;j+=step){
22         int k=j+mh;
23         std::complex<T> u=out[j],t=wi*out[
24             k];
25         out[j]=u+t;
26         out[k]=u-t;
27     }
28 }
29 if(is_inv)for(int i=0;i<N;++i)out[i]/=N;
30 }

```

7.9 find_real_root.cpp

```

1 // an*x^n + ... + a1x + a0 = 0;
2 int sign(double x){
3     return x < -eps ? -1 : x > eps;
4 }
5 double get(const vector<double>&coef, double
6     x){
7     double e = 1, s = 0;
8     for(auto i : coef) s += i*e, e *= x;
9     return s;
10 }
11 double find(const vector<double>&coef, int n
12     , double lo, double hi){
13     double sign_lo, sign_hi;
14     if( !(sign_lo = sign(get(coef,lo))) )
15         return lo;
16     if( !(sign_hi = sign(get(coef,hi))) )
17         return hi;
18     if(sign_lo * sign_hi > 0) return INF;
19     for(int stp = 0; stp < 100 && hi - lo >
20         eps; ++stp){
21         double m = (lo+hi)/2.0;
22         int sign_mid = sign(get(coef,m));
23         if(!sign_mid) return m;
24         if(sign_lo*sign_mid < 0) hi = m;
25         else lo = m;
26     }
27     return (lo+hi)/2.0;
28 }
29 vector<double> cal(vector<double>coef, int n
30 ) {
31     vector<double>res;
32     if(n == 1){
33         if(sign(coef[1])) res.pb(-coef[0]/
34             coef[1]);
35         return res;
36     }
37     vector<double>dcoef(n);
38     for(int i = 0; i < n; ++i) dcoef[i] =
39         coef[i+1]*(i+1);
40     vector<double>droot= cal(dcoef, n-1);
41     droot.insert(droot.begin(), -INF);

```

```

37 droot.pb(INF);
38 for(int i = 0; i+1 < droot.size(); ++i){
39     double tmp = find(coef, n, droot[i],
40         droot[i+1]);
41     if(tmp < INF) res.pb(tmp);
42 }
43 return res;
44 }
45 int main () {
46     vector<double>ve;
47     vector<double>ans = cal(ve, n);
48     // 視情況把答案 +eps · 避免 -0
49 }

```

7.10 formula.tex

$$\sum_{d|n} \phi(n) = n$$

$$\sum_{d|n} \mu(n) = (n == 1)$$

$$g(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) * g(n/d)$$

$$\text{Catalan number} : (2n)!/n!/n!/(n+1)$$

$$\text{Harmonic series } H_n = \ln(n) + \gamma + 1/(2n) - 1/(12n) + 1/(120n^3) - \dots$$

$$\gamma = 0.57721566490153286060651209008240243104215\dots$$

$$i - \text{th gray code} : i^{(i>1)}$$

$$SG(A+B) = SG(A) \oplus SG(B)$$

7.11 Gauss_Elimination.cpp

```

1 const int MAX = 300;
2 const double EPS = 1e-8;
3 double mat[MAX][MAX];
4 void Gauss(int n) {
5     for(int i=0; i<n; i++) {
6         bool ok = 0;
7         for(int j=i; j<n; j++) {
8             if(fabs(mat[j][i]) > EPS) {
9                 swap(mat[j], mat[i]);
10                ok = 1;
11                break;
12            }
13        }
14        if(!ok) continue;
15        double fs = mat[i][i];
16        for(int j=i+1; j<n; j++) {
17            double r = mat[j][i] / fs;
18            for(int k=i; k<n; k++) {
19                mat[j][k] -= mat[i][k] * r;
20            }
21        }
22    }
23 }
24 }
25 }

```

7.12 LinearCongruence.cpp

```

1 pair<LL,LL> LinearCongruence(LL a[],LL b[],
  LL m[],int n) {
2   // a[i]*x = b[i] (mod m[i])
3   for(int i=0;i<n;++i) {
4     LL x, y, d = extgcd(a[i],m[i],x,y);
5     if(b[i]%d!=0) return make_pair(-1LL
6       ,0LL);
7     m[i] /= d;
8     b[i] = Llmul(b[i]/d,x,m[i]);
9   }
10  LL lastb = b[0], lastm = m[0];
11  for(int i=1;i<n;++i) {
12    LL x, y, d = extgcd(m[i],lastm,x,y);
13    if((lastb-b[i])%d!=0) return
14      make_pair(-1LL,0LL);
15    lastb = Llmul((lastb-b[i])/d,x,
16      lastm/d)*m[i];
17    lastm = (lastm/d)*m[i];
18    lastb = (lastb+b[i])%lastm;
19  }
20  return make_pair(lastb<0?lastb+lastm:
21    lastb,lastm);
22 }

```

7.13 Lucas.cpp

```

1 int mod_fact(int n,int &e){
2   e=0;
3   if(n==0)return 1;
4   // (n/p)! % p
5   int res=mod_fact(n/P,e);
6   e += n/P;
7   if( (n/P) %2 == 0){// = 1
8     return res*fact[n%P]%P;
9   }
10  // = -1
11  return res*(P-fact[n%P])%P;
12 }
13 int extGCD(int a,int b,int &x,int &y){
14  int d=a;
15  if(b!=0){
16    d=extGCD(b,a%b,y,x);
17    y -= (a/b)*x;
18  }else{
19    x=1;y=0;
20  }
21  return d;
22 }
23 int modInverse(int n){
24  int x,y;
25  extGCD(n,P,x,y);
26  return (P+x%P)%P;
27 }
28 int Cmod(int n,int m){
29  int a1,a2,a3,e1,e2,e3;
30  a1=mod_fact(n,e1);
31  a2=mod_fact(m,e2);
32  a3=mod_fact(n-m,e3);
33  if(e1>e2+e3)return 0;
34  return a1*modInverse(a2*a3%P)%P;
35 }

```

7.14 NTT.cpp

```

1 2615053605667*(2^18)+1,3
2 15*(2^27)+1,31
3 479*(2^21)+1,3
4 7*17*(2^23)+1,3
5 3*3*211*(2^19)+1,5
6 25*(2^22)+1,3
7 template<typename T,typename VT=std::vector<
  T>>
8 struct NTT{
9   const T P,G;
10  NTT(T p=(1<<23)*7*17+1,T g=3):P(p),G(g){}
11  unsigned int bit_reverse(unsigned int a,
12    int len){
13    a=((a&0x55555555U)<<1)|((a&0xAAAAAAAAU)
14      >>1);
15    a=((a&0x33333333U)<<2)|((a&0xCCCCCCC0U)
16      >>2);
17    a=((a&0x0F0F0F0FU)<<4)|((a&0xF0F0F0F0U)
18      >>4);
19    a=((a&0x00FF00FFU)<<8)|((a&0xFF00FF00U)
20      >>8);
21    a=((a&0x0000FFFFU)<<16)|((a&0xFFFF0000U)
22      >>16);
23    return a>>(32-len);
24  }
25  T pow_mod(T n,T k,T m){
26    T ans=1;
27    for(n=(n>=m?n%m:n);k;k>>=1){
28      if(k&1)ans=ans*n%m;
29      n=n*n%m;
30    }
31    return ans;
32  }
33  void ntt(bool is_inv,VT &in,VT &out,int N)
34  {
35    int bitlen=std::__lg(N);
36    for(int i=0;i<N;++i)out[bit_reverse(i,
37      bitlen)]=in[i];
38    for(int step=2,id=1;step<=N;step<=1,++
39      id){
40      T wn=pow_mod(G,(P-1)>>id,P),wi=1,u,t;
41      const int mh=step>>1;
42      for(int i=0;i<mh;++i){
43        for(int j=i;j<N;j+=step){
44          u=out[j],t=wi*out[j+mh]%P;
45          out[j]=u+t;
46          out[j+mh]=u-t;
47          if(out[j]>=P)out[j]-=P;
48          if(out[j+mh]<0)out[j+mh]+=P;
49        }
50        wi=wi*wn%P;
51      }
52    }
53    if(is_inv){
54      for(int i=1;i<N/2;++i)std::swap(out[i]
55        ,out[N-i]);
56    }
57    T invn=pow_mod(N,P-2,P);
58    for(int i=0;i<N;++i)out[i]=out[i]*invn
59      %P;
60  }
61 }
62 };

```

7.15 random.cpp

```

1 inline int random_int(){
2   static int seed=20160424;
3   return seed+=(seed<<16)+0x1db3d743;
4 }
5 inline long long random_long_long(){
6   static long long seed=20160424;
7   return seed+=(seed<<32)+0xdb3d742c265539d;
8 }

```

7.16 外星模運算.cpp

```

1 //a[0]^a[1]^a[2]^...
2 #include<bits/stdc++.h>
3 using namespace std;
4 #define maxn 1000000
5 int euler[maxn+5];
6 bool is_prime[maxn+5];
7 inline void init_euler(){
8   is_prime[1]=1;//一不是質數
9   for(int i=1;i<=maxn;i++)euler[i]=i;
10  for(int i=2;i<=maxn;i++){
11    if(!is_prime[i]){//是質數
12      euler[i]--;
13      for(int j=i<<1;j<=maxn;j+=i){
14        is_prime[j]=1;
15        euler[j]=euler[j]/i*(i-1);
16      }
17    }
18  }
19 }
20 inline long long pow(long long a,long long b
21   ,long long mod){//a^b%mod
22   long long ans=1;
23   for(;b;a=a*a%mod,b>>=1)
24     if(b&1)ans=ans*a%mod;
25   return ans;
26 }
27 bool isless(long long *a,int n,int k){
28   if(*a==1)return k>1;
29   if(--n==0)return *a<k;
30   int next=0;
31   for(long long b=1;b<k;++next)
32     b*=*a;
33   return isless(a+1,n,next);
34 }
35 long long high_pow(long long *a,int n,long
36   long mod){
37   if(*a==1||--n==0)return *a%mod;
38   int k=0,r=euler[mod];
39   for(long long tma=1;tma!=pow(*a,k+r,mod)
40     ;++k)
41     tma=tma*(*a)%mod;
42   if(isless(a+1,n,k))return pow(*a,high_pow(
43     a+1,n,k),mod);
44   int tmd=high_pow(a+1,n,r);
45   int t=(tmd-k+r)%r;
46   return pow(*a,k+t,mod);
47 }
48 }
49 long long a[1000005];

```

```

45 int t,mod;
46 int main(){
47   init_euler();
48   scanf("%d",&t);
49   #define n 4
50   while(t--){
51     for(int i=0;i<n;++i)scanf("%Lld",&a[i]);
52     scanf("%d",&mod);
53     printf("%Lld\n",high_pow(a,n,mod));
54   }
55   return 0;
56 }

```

7.17 模運算模板.cpp

```

1 template<typename T,long long mod>
2 struct mod_t{//mod只能是質數
3   T data;
4   mod_t(){}
5   mod_t(const T &d):data((d%mod+mod)%mod){}
6   mod_t pow(T b)const{
7     mod_t ans(1);
8     for(mod_t now=*this;b;now=now*b,b/=2)
9       if(b%2)ans=ans*now;
10    return ans;
11  }
12  mod_t operator-(int)const{
13    return mod_t(mod-data);
14  }
15  mod_t operator+(const mod_t &b)const{
16    return mod_t((data+b.data)%mod);
17  }
18  mod_t operator-(const mod_t &b)const{
19    return mod_t((data-b.data+mod)%mod);
20  }
21  mod_t operator*(const mod_t &b)const{
22    return mod_t((data*b.data)%mod);
23  }
24  mod_t operator/(const mod_t &b)const{
25    return *this*b.pow(mod-2);//*this *
26    Inverse(b)
27  }
28  operator T()const{return data;}
29  friend istream &operator>>(istream &i,
30    mod_t &b){
31    T d;
32    i>>d;
33    b=mod_t(d);
34    return i;
35  }
36 };

```

8 String

8.1 AC 自動機.cpp

```

1 template<char L='a',char R='z'>
2 class ac_automaton{
3 private:
4     struct joe{
5         int next[R-L+1], fail, efl, ed, cnt_dp, vis;
6     };
7     joe() : ed(0), cnt_dp(0), vis(0){
8         for(int i=0; i<=R-L; ++i) next[i]=0;
9     };
10 public:
11     std::vector<joe> S;
12     std::vector<int> q;
13     int qs, qe, vt;
14     ac_automaton() : S(1), qs(0), qe(0), vt(0){
15         void clear(){
16             q.clear();
17             S.resize(1);
18             for(int i=0; i<=R-L; ++i) S[0].next[i]=0;
19             S[0].cnt_dp=S[0].vis=qs=qe=vt=0;
20         }
21         void insert(const char *s){
22             int o=0;
23             for(int i=0; id=s[i]-L; ++i){
24                 if(!S[o].next[id]){
25                     S.push_back(joe());
26                     S[o].next[id]=S.size()-1;
27                 }
28                 o=S[o].next[id];
29                 ++S[o].ed;
30             }
31             ++S[o].ed;
32         }
33         void build_fail(){
34             S[0].fail=S[0].efl=-1;
35             q.clear();
36             q.push_back(0);
37             ++qe;
38             while(qs!=qe){
39                 int pa=q[qs++], id, t;
40                 for(int i=0; i<=R-L; ++i){
41                     t=S[pa].next[i];
42                     if(!t) continue;
43                     id=S[pa].fail;
44                     while(~id&&S[id].next[i]) id=S[id].fail;
45                     S[t].fail=id;
46                     S[t].efl=S[t].fail;
47                     S[t].ed=S[t].fail;
48                     q.push_back(t);
49                     ++qe;
50                 }
51             }
52             /*DP出每個前綴在字串s出現的次數並傳回所有字串被s匹配成功的次數O(N*M)*/
53             int match_0(const char *s){
54                 int ans=0, id, p=0, i;
55                 for(i=0; s[i]; ++i){
56                     id=s[i]-L;
57                     while(!S[p].next[id]&&p=S[p].fail;
58                     if(!S[p].next[id]) continue;
59                     p=S[p].next[id];
60                     ++S[p].cnt_dp; /*匹配成功則它所有後綴都可以被匹配(DP計算)*/
61                 }
62                 for(i=qe-1; i>=0; --i){
63                     ans+=S[q[i]].cnt_dp*S[q[i]].ed;
64                     if(~S[q[i]].fail) S[q[i]].fail;
65                     cnt_dp+=S[q[i]].cnt_dp;
66                 }
67                 return ans;
68             }
69             /*多串匹配走efl邊並傳回所有字串被s匹配成功的次數O(N*M^1.5)*/
70             int match_1(const char *s) const{
71                 int ans=0, id, p=0, t;
72                 for(int i=0; s[i]; ++i){
73                     id=s[i]-L;
74                     while(!S[p].next[id]&&p=S[p].fail;
75                     if(!S[p].next[id]) continue;
76                     p=S[p].next[id];
77                     if(S[p].ed) ans+=S[p].ed;
78                     for(t=S[p].efl; ~t; t=S[t].efl){
79                         ans+=S[t].ed; /*因為都走efl邊所以保證匹配成功*/
80                     }
81                 }
82                 return ans;
83             }
84             /*枚舉(s的子字串nA)的所有相異字串各恰一次並傳回次數O(N*M^(1/3))*/
85             int match_2(const char *s){
86                 int ans=0, id, p=0, t;
87                 ++vt;
88                 /*把戳記vt+=1, 只要vt沒溢位, 所有S[p].vis==vt就會變成false
89                 這種利用vt的方法可以O(1)歸零vis陣列*/
90                 for(int i=0; s[i]; ++i){
91                     id=s[i]-L;
92                     while(!S[p].next[id]&&p=S[p].fail;
93                     if(!S[p].next[id]) continue;
94                     p=S[p].next[id];
95                     if(S[p].ed&&S[p].vis!=vt){
96                         S[p].vis=vt;
97                         ans+=S[p].ed;
98                     }
99                     for(t=S[p].efl; ~t&&S[t].vis!=vt; t=S[t].efl){
100                         S[t].vis=vt;
101                         ans+=S[t].ed; /*因為都走efl邊所以保證匹配成功*/
102                     }
103                 }
104                 return ans;
105             }
106             /*把AC自動機變成真的自動機*/
107             void evolution(){
108                 for(qs=1; qs!=qe; ++q){
109                     int p=q[qs++];
110                     for(int i=0; i<=R-L; ++i)
111                         if(S[p].next[i]==0) S[p].next[i]=S[p].fail.next[i];
112                 }
113             }
114         }

```

8.2 hash.cpp

```

1 #define MAXN 1000000
2 #define prime_mod 1073676287
3 /*prime_mod 必須要要是質數*/
4 typedef long long T;
5 char s[MAXN+5];
6 T h[MAXN+5]; /*hash陣列*/
7 T h_base[MAXN+5]; /*h_base[n]=(prime^n)%prime_mod*/
8 inline void hash_init(int len, T prime=0)
9     xdefaced(){
10         h_base[0]=1;
11         for(int i=1; i<=len; ++i){
12             h[i]=(h[i-1]*prime+s[i-1])%prime_mod;
13             h_base[i]=(h_base[i-1]*prime)%prime_mod;
14         }
15     }
16 inline T get_hash(int l, int r){ /*閉區間寫法, 設編號為0 ~ len-1*/
17     return (h[r+1]-(h[l]*h_base[r-l+1])%prime_mod+prime_mod)%prime_mod;

```

8.3 KMP.cpp

```

1 /*產生fail function*/
2 inline void kmp_fail(char *s, int len, int *fail){
3     int id=-1;
4     fail[0]=-1;
5     for(int i=1; i<len; ++i){
6         while(~id&&S[id+1]!=s[i]) id=fail[id];
7         if(s[id+1]==s[i]) ++id;
8         fail[i]=id;
9     }
10 }
11 /*以字串B匹配字串A, 傳回匹配成功的數量(用B的fail)*/
12 inline int kmp_match(char *A, int lenA, char *B, int lenB, int *fail){
13     int id=-1, ans=0;
14     for(int i=0; i<lenA; ++i){
15         while(~id&&B[id+1]!=A[i]) id=fail[id];
16         if(B[id+1]==A[i]) ++id;
17         if(id==lenB-1){ /*匹配成功*/
18             ++ans;
19             id=fail[id];
20         }
21     }
22     return ans;
23 }

```

8.4 manacher.cpp

```

1 //原字串: asdsasdsa
2 //先把字串變成這樣: @a#s#d#s#a#s#d#s#a#

```

```

3 inline void manacher(char *s, int len, int *z)
4 {
5     int l=0, r=0;
6     for(int i=1; i<len; ++i){
7         z[i]=r>i?min(z[2*i-l-i], r-i):1;
8         while(s[i+z[i]]==s[i-z[i]]) ++z[i];
9         if(z[i]+i>r) r=z[i]+i, l=i;
10 }

```

8.5 minimal_string_rotation.cpp

```

1 int min_string_rotation(const string &s){
2     int n=s.size(), i=0, j=1, k=0;
3     while(i<n&&j<n&&k<n){
4         int t=s[(i+k)%n]-s[(j+k)%n];
5         ++k;
6         if(t){
7             if(t>0) i+=k;
8             else j+=k;
9             if(i==j) ++j;
10            k=0;
11        }
12    }
13    return min(i, j); //傳回最小循環表示法起始位置
14 }

```

8.6 suffix_array_lcp.cpp

```

1 #define radix_sort(x,y){\
2     for(i=0; i<A; ++i) c[i]=0;\
3     for(i=0; i<len; ++i) c[x[y[i]]]++;\
4     for(i=1; i<A; ++i) c[i]+=c[i-1];\
5     for(i=len-1; i>=0; --i) sa[--c[x[y[i]]]]=y[i];\
6 }
7 void suffix_array(const char *s, int len, int *sa, int *rank, int *tmp, int *c){
8     int A='z'+1, i, k, id, *t;
9     for(i=0; i<len; ++i){
10         tmp[i]=i;
11         rank[i]=s[i];
12     }
13     radix_sort(rank, tmp);
14     for(k=1; id<len-1; k<=1){
15         id=0;
16         for(i=len-k; i<len; ++i) tmp[id++] = i;
17         for(i=0; i<len; ++i){
18             if(sa[i]>=k) tmp[id++] = sa[i]-k;
19         }
20         radix_sort(rank, tmp);
21         t=rank; rank=tmp; tmp=t;
22         id=0;
23         rank[sa[0]]=0;
24         for(i=1; i<len; ++i){
25             if(tmp[sa[i-1]]!=tmp[sa[i]] || sa[i-1]+k>len || tmp[sa[i-1]+k]!=tmp[sa[i]+k]) ++id;

```



```

26     rank[sa[i]]=id;
27 }
28 A=id+1;
29 }
30 }
31 #undef radix_sort
32 //h:高度數組 sa:後綴數組 rank:排名
33 inline void suffix_array_lcp(const char *s,
34     int len,int *h,int *sa,int *rank){
35     for(int i=0;i<len;++i)rank[sa[i]]=i;
36     for(int i=0,k=0;i<len;++i){
37         if(rank[i]==0)continue;
38         if(k)--k;
39         while(s[i+k]==s[sa[rank[i]-1]+k])++k;
40         h[rank[i]]=k;
41     }
42 }

```

8.7 Z.cpp

```

1 inline void z_alg(char *s,int len,int *z){
2     int l=0,r=0;
3     z[0]=len;
4     for(int i=1;i<len;++i){
5         z[i]=i>r?0:(i-l+z[i-l]<z[l]?z[i-l]:r-i
6             +1);
7         while(i+z[i]<len&&s[i+z[i]]==s[z[i]]++)z[i]++;
8         if(i+z[i]-1>r)r=i+z[i]-1,l=i;
9     }
}

```

9 Tarjan

9.1 dominator_tree.cpp

```

1 struct dominator_tree{
2     static const int MAXN=5005;
3     int n;// 1-base
4     vector<int> suc[MAXN],pre[MAXN]; //存圖和反
5     //向圖
6     int fa[MAXN],dfn[MAXN],id[MAXN],Time;//for
7     //dfs
8     int semi[MAXN],idom[MAXN];
9     int anc[MAXN],best[MAXN]; //disjoint set
10    vector<int> dom[MAXN]; //dominator_tree存這
11    //裡
12    void init(int _n){
13        n=_n;
14        for(int i=1;i<=n;++i)suc[i].clear(),pre[i].clear();
15    }
16    void add_edge(int u,int v){
17        suc[u].push_back(v);
18        pre[v].push_back(u);
19    }
}

```

```

17 void dfs(int u){
18     dfn[u]=++Time,id[Time]=u;
19     for(auto v:suc[u]){
20         if(dfn[v])continue;
21         dfs(v),fa[dfn[v]]=dfn[u];
22     }
23 }
24 int find(int x){
25     if(x==anc[x])return x;
26     int y=find(anc[x]);
27     if(semi[best[x]]>semi[best[anc[x]]])best[x]=best[anc[x]];
28     return anc[x]=y;
29 }
30 void tarjan(int r){
31     Time=0;
32     for(int t=1;t<=n;++t){
33         dfn[t]=idom[t]=0;//u=r或是u無法到達r時
34         //idom[id[u]]=0
35         dom[t].clear();
36         anc[t]=best[t]=semi[t]=t;
37     }
38     dfs(r);
39     for(int y=Time;y>=2;--y){
40         int x=fa[y],idy=id[y];
41         for(auto z:pre[idy]){
42             if(!(z=dfn[z]))continue;
43             find(z);
44             semi[y]=min(semi[y],semi[best[z]]);
45         }
46         dom[semi[y]].push_back(y);
47         anc[y]=x;
48         for(auto z:dom[x]){
49             find(z);
50             idom[z]=semi[best[z]]<x?best[z]:x;
51         }
52         dom[x].clear();
53     }
54     for(int u=2;u<=Time;++u){
55         if(idom[u]!=semi[u])idom[u]=idom[idom[u]];
56         dom[id[idom[u]]].push_back(id[u]);
57     }
58 }dom;

```

9.2 tnfsb017_2_sat.cpp

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define MAXN 8001
4 #define MAXN2 MAXN*4
5 #define n(X) ((X)+2*MAXN)
6 vector<int> v[MAXN2];
7 vector<int> rv[MAXN2];
8 vector<int> vis_t;
9 int N,M;
10 void addedge(int s,int e){
11     v[s].push_back(e);
12     rv[e].push_back(s);
13 }
14 int scc[MAXN2];
15 bool vis[MAXN2]={false};

```

```

16 void dfs(vector<int> *uv,int n,int k=-1){
17     vis[n]=true;
18     for(int i=0;i<uv[n].size();++i)
19         if(!vis[uv[n][i]])
20             dfs(uv,uv[n][i],k);
21     if(uv==v)vis_t.push_back(n);
22     scc[n]=k;
23 }
24 void solve(){
25     for(int i=1;i<=N;++i){
26         if(!vis[i])dfs(v,i);
27         if(!vis[n(i)])dfs(v,n(i));
28     }
29     memset(vis,0,sizeof(vis));
30     int c=0;
31     for(int i=vis_t.size()-1;i>=0;--i)
32         if(!vis[vis_t[i]])
33             dfs(rv,vis_t[i],c++);
34 }
35 int main(){
36     int a,b;
37     scanf("%d%d",&N,&M);
38     for(int i=1;i<=N;++i){
39         // (A or B)&(A & !B) A^B
40         a=i*2-1;
41         b=i*2;
42         addedge(n(a),b);
43         addedge(n(b),a);
44         addedge(a,n(b));
45         addedge(b,n(a));
46     }
47     while(M--){
48         scanf("%d%d",&a,&b);
49         a = a>0?a*2-1:-a*2;
50         b = b>0?b*2-1:-b*2;
51         // A or B
52         addedge(n(a),b);
53         addedge(n(b),a);
54     }
55     solve();
56     bool check=true;
57     for(int i=1;i<=2*N;++i)
58         if(scc[i]==scc[n(i)])
59             check=false;
60     if(check){
61         printf("%d\n",N);
62         for(int i=1;i<=2*N;i+=2){
63             if(scc[i]>scc[i+2*N])
64                 putchar('+');
65             else
66                 putchar('-');
67         }
68         putchar('\n');
69     }else puts("0");
70     return 0;
71 }

```

9.3 橋連通分量.cpp

```

1 #define N 1005
2 struct edge{
3     int u,v;
4     bool is_bridge;

```

```

5     edge(int u=0,int v=0):u(u),v(v),is_bridge(0){}
6 };
7 vector<edge> E;
8 vector<int> G[N]; // 1-base
9 int low[N],vis[N],Time;
10 int bcc_id[N],bridge_cnt,bcc_cnt; // 1-base
11 int st[N],top; // BCC用
12 inline void add_edge(int u,int v){
13     G[u].push_back(E.size());
14     E.push_back(edge(u,v));
15     G[v].push_back(E.size());
16     E.push_back(edge(v,u));
17 }
18 void dfs(int u,int re=-1){ //u當前點·re為u連
19     //接前一個點的邊
20     int v;
21     low[u]=vis[u]=++Time;
22     st[top++]=u;
23     for(size_t i=0;i<G[u].size();++i){
24         int e=G[u][i];v=E[e].v;
25         if(!vis[v]){
26             dfs(v,e^1); //e^1反向邊
27             low[u]=min(low[u],low[v]);
28             if(vis[u]<low[v]){
29                 E[e].is_bridge=E[e^1].is_bridge=1;
30                 ++bridge_cnt;
31             }else if(vis[v]<vis[u]&&e!=re)
32                 low[u]=min(low[u],vis[v]);
33         }
34         if(vis[u]==low[u]){ //處理BCC
35             ++bcc_cnt; // 1-base
36             do bcc_id[v]=st[--top]=bcc_cnt; //每個點
37                 //所在的BCC
38             while(v!=u);
39         }
40     }
41     inline void bcc_init(int n){
42         Time=bcc_cnt=bridge_cnt=top=0;
43         E.clear();
44         for(int i=1;i<=n;++i){
45             G[i].clear();
46             vis[i]=bcc_id[i]=0;
47         }
}

```

9.4 雙連通分量 & 割點.cpp

```

1 #define N 1005
2 vector<int> G[N]; // 1-base
3 vector<int> bcc[N]; //存每塊雙連通分量的點
4 int low[N],vis[N],Time;
5 int bcc_id[N],bcc_cnt; // 1-base
6 bool is_cut[N]; //是否為割點·割點的bcc_id沒
7 //意義
8 int st[N],top;
9 void dfs(int u,int pa=-1){ //u當前點·pa父親
10     int v,child=0;
11     low[u]=vis[u]=++Time;
12     st[top++]=u;

```

```

12 for(size_t i=0;i<G[u].size();++i){
13     if(!vis[v=G[u][i]]){
14         dfs(v,u,++child;
15         low[u]=min(low[u],low[v]);
16         if(vis[u]<=low[v]){
17             is_cut[u]=1;
18             bcc[++bcc_cnt].clear();
19             int t;
20             do{
21                 bcc_id[t=st[--top]]=bcc_cnt;
22                 bcc[bcc_cnt].push_back(t);
23             }while(t!=v);
24             bcc_id[u]=bcc_cnt;
25             bcc[bcc_cnt].push_back(u);
26         }
27     }else if(vis[v]<vis[u]&&v!=pa)//反向邊
28         low[u]=min(low[u],vis[v]);
29 }
30 if(pa==--1&&child<2)is_cut[u]=0;//u是dfs樹
31     的根要特判
32 inline void bcc_init(int n){
33     Time=bcc_cnt=top=0;
34     for(int i=1;i<=n;++i){
35         G[i].clear();
36         is_cut[i]=vis[i]=bcc_id[i]=0;
37     }
38 }

```

10 Tree_problem

10.1 HeavyLight.cpp

```

1 #include<vector>
2 #define MAXN 100005
3 typedef std::vector<int>::iterator VIT;
4 int siz[MAXN],max_son[MAXN],pa[MAXN],dep[
5     MAXN];
6 /*節點大小、大小最大的孩子、父母節點、深度*/
7 int link_top[MAXN],link[MAXN],cnt;
8 /*每個點所在鍊的鏈頭、樹鏈剖分的DFS序、時間
9     戳*/
10 std::vector<int> >G[MAXN];/*用vector存樹*/
11 void find_max_son(int x){
12     siz[x]=1;
13     max_son[x]=-1;
14     for(VIT i=G[x].begin();i!=G[x].end();++i){
15         if(*i==pa[x])continue;
16         pa[*i]=x;
17         dep[*i]=dep[x]+1;
18         find_max_son(*i);
19         if(max_son[x]==-1||siz[*i]>siz[max_son[x]
20             ])max_son[x]=*i;
21         siz[x]+=siz[*i];
22     }
23 }
24 void build_link(int x,int top){
25     link[x]=++cnt;/*記錄x點的時間戳*/
26     link_top[x]=top;
27     if(max_son[x]==-1)return;

```

```

25     build_link(max_son[x],top);/*優先走訪最大
26         孩子*/
27     for(VIT i=G[x].begin();i!=G[x].end();++i){
28         if(*i==max_son[x]||*i==pa[x])continue;
29         build_link(*i,*i);
30     }
31     inline int find_lca(int a,int b){
32         /*求LCA·可以在過程中對區間進行處理*/
33         int ta=link_top[a],tb=link_top[b];
34         while(ta!=tb){
35             if(dep[ta]<dep[tb]){
36                 std::swap(ta,tb);
37                 std::swap(a,b);
38             }
39             /*這裡可以對a所在的鏈做區間處理
40                 區間為(Link[ta],Link[a])
41                 ta=link_top[a=pa[ta]];
42             */
43         }
44         /*最後a,b會在同一條鏈·若a!=b還要在進行一
45             次區間處理*/
46         return dep[a]<dep[b]?a:b;

```

10.2 LCA.cpp

```

1 #define MAXN 100000
2 #define MAX_LOG 17
3 int pa[MAX_LOG+1][MAXN+5];
4 int dep[MAXN+5];
5 vector<int>G[MAXN+5];
6 void dfs(int x,int p){/*dfs(1,-1);
7     pa[0][x]=p;
8     for(int i=0;i+1<MAX_LOG;++i)pa[i+1][x]=pa[
9         i][pa[i][x]];
10     for(auto &i:G[x]){
11         if(i==p)continue;
12         dep[i]=dep[x]+1;
13         dfs(i,x);
14     }
15     inline int jump(int x,int d){
16         for(int i=0;i<d;++i)if((x>>i)&1)x=pa[k][x
17             ];
18         return x;
19     }
20     inline int find_lca(int a,int b){
21         if(dep[a]>dep[b])swap(a,b);
22         b=jump(b,dep[b]-dep[a]);
23         if(a==b)return a;
24         for(int i=MAX_LOG;i>=0;--i){
25             if(pa[i][a]!=pa[i][b]){
26                 a=pa[i][a];
27                 b=pa[i][b];
28             }
29         }
30         return pa[0][a];

```

10.3 link_cut_tree.cpp

```

1 #include<vector>
2 struct splay_tree{
3     int ch[2],pa;/*子節點跟父母*/
4     bool rev;/*反轉的懶惰標記*/
5     splay_tree():pa(0),rev(0){ch[0]=ch[1]=0;}
6 };
7 std::vector<splay_tree> node;
8 /*
9     有的時候用vector會TLE·要注意
10     這邊以node[0]作為null節點
11 */
12 inline bool isroot(int x){/*判斷是否為這棵
13     splay tree的根*/
14     return node[node[x].pa].ch[0]!=x&&node[
15         node[x].pa].ch[1]!=x;
16 }
17 inline void down(int x){/*懶惰標記下推*/
18     if(node[x].rev){
19         if(node[x].ch[0])node[node[x].ch[0]].rev
20             ^=1;
21         if(node[x].ch[1])node[node[x].ch[1]].rev
22             ^=1;
23         std::swap(node[x].ch[0],node[x].ch[1]);
24         node[x].rev^=1;
25     }
26 }
27 void push_down(int x){/*將所有祖先的懶惰標記
28     下推*/
29     if(!isroot(x))push_down(node[x].pa);
30     down(x);
31 }
32 inline void up(int x){/*將子節點的資訊向上
33     更新*/
34     inline void rotate(int x){/*旋轉·會自行判斷
35         轉的方向*/
36         int y=node[x].pa,z=node[y].pa,d=(node[y].
37             ch[1]==x);
38         node[x].pa=z;
39         if(!isroot(y))node[z].ch[node[z].ch[1]==y
40             ]=x;
41         node[y].ch[d]=node[x].ch[d^1];
42         node[node[y].ch[d]].pa=y;
43         node[y].pa=x,node[x].ch[d^1]=y;
44         up(y);
45         up(x);
46     }
47     inline void splay(int x){/*將節點x伸展到所在
48         splay tree的根*/
49         push_down(x);
50         while(!isroot(x)){
51             int y=node[x].pa;
52             if(!isroot(y)){
53                 int z=node[y].pa;
54                 if((node[z].ch[0]==y)^(node[y].ch[0]==
55                     x))rotate(y);
56                 else rotate(x);
57             }
58             rotate(x);
59         }
60     }
61     inline int access(int x){

```

```

51     int last=0;
52     while(x){
53         splay(x);
54         node[x].ch[1]=last;
55         up(x);
56         last=x;
57         x=node[x].pa;
58     }
59     return last;/*回傳access後splay tree的根*/
60 }
61 inline void access(int x,bool is=0){/*is=0就
62     是一般的access*/
63     int last=0;
64     while(x){
65         splay(x);
66         if(is&&!node[x].pa){
67             //printf("%d\n",max(node[last].ma,node
68                 [node[x].ch[1]].ma));
69         }
70         node[x].ch[1]=last;
71         up(x);
72         last=x;
73         x=node[x].pa;
74     }
75     inline void query_edge(int u,int v){
76         access(u);
77         access(v,1);
78     }
79     inline void make_root(int x){
80         access(x),splay(x);
81         node[x].rev^=1;
82     }
83     inline void make_root(int x){
84         node[access(x)].rev^=1;
85         splay(x);
86     }
87     inline void cut(int x,int y){
88         make_root(x);
89         access(y);
90         splay(y);
91         node[y].ch[0]=0;
92         node[x].pa=0;
93     }
94     inline void cut_parents(int x){
95         access(x);
96         splay(x);
97         node[node[x].ch[0]].pa=0;
98         node[x].ch[0]=0;
99     }
100     inline void link(int x,int y){
101         make_root(x);
102         node[x].pa=y;
103     }
104     inline int find_root(int x){
105         x=access(x);
106         while(node[x].ch[0])x=node[x].ch[0];
107         splay(x);
108         return x;
109     }
110     inline int query(int u,int v){
111         /*
112         傳回uv路徑splay tree的根結點
113         這種寫法無法求LCA
114         */

```

```

114 make_root(u);
115 return access(v);
116 }
117 inline int query_lca(int u,int v){
118 /*假設求鏈上點權的總和，sum是子樹的權重和，
119 data是節點的權重*/
120 access(u);
121 int lca=access(v);
122 splay(u);
123 if(u==lca){
124 //return node[lca].data+node[node[lca].
125 ch[1]].sum;
126 }else{
127 //return node[lca].data+node[node[lca].
128 ch[1]].sum+node[u].sum;
129 }
130 }
131 struct EDGE{
132 int a,b,w;
133 }e[10005];
134 int n;
135 std::vector<std::pair<int,int>> >G[10005];
136 /*first表示子節點，second表示邊的編號*/
137 int pa[10005],edge_node[10005];
138 /*pa是父母節點，暫存用的，edge_node是每個編
139 被存在哪個點裡面的陣列*/
140 inline void bfs(int root){
141 /*在建構的時候把每個點都設成一個splay tree，
142 不會壞掉*/
143 std::queue<int> q;
144 for(int i=1;i<=n;++i)pa[i]=0;
145 q.push(root);
146 while(q.size()){
147 int u=q.front();
148 q.pop();
149 for(int i=0;i<(int)G[u].size();++i){
150 int v=G[u][i].first;
151 if(v!=pa[u]){
152 pa[v]=u;
153 node[v].pa=u;
154 node[v].data=e[G[u][i].second].w;
155 edge_node[G[u][i].second]=v;
156 up(v);
157 q.push(v);
158 }
159 }
160 }
161 }
162 inline void change(int x,int b){
163 splay(x);
164 //node[x].data=b;
165 up(x);
166 }
167 }
168
169 bool vis[MAXN];
170 inline void init(){
171 for(int i=0;i<=n;++i){
172 g[i].clear();
173 vis[i]=0;
174 }
175 }
176 void get_dis(vector<int> &dis,int u,int pa,
177 int d){
178 dis.push_back(d);
179 for(size_t i=0;i<g[u].size();++i){
180 int v=g[u][i].first,w=g[u][i].second;
181 if(v!=pa&&!vis[v])get_dis(dis,v,u,d+w);
182 }
183 }
184 vector<int> dis;//這東西如果放在函數裡會TLE
185 int cal(int u,int d){
186 dis.clear();
187 get_dis(dis,u,-1,d);
188 sort(dis.begin(),dis.end());
189 int l=0,r=dis.size()-1,res=0;
190 while(l<r){
191 while(l<r&&dis[l]+dis[r]>k)--r;
192 res+=r-(l+1);
193 }
194 return res;
195 }
196 pair<int,int> tree_centroid(int u,int pa,
197 const int sz){
198 size[u]=1;//找樹重心，second是重心
199 pair<int,int> res(INT_MAX,-1);
200 int ma=0;
201 for(size_t i=0;i<g[u].size();++i){
202 int v=g[u][i].first;
203 if(v==pa||vis[v])continue;
204 res=min(res,tree_centroid(v,u,sz));
205 size[u]+=size[v];
206 ma=max(ma,size[v]);
207 }
208 ma=max(ma,sz-size[u]);
209 return min(res,make_pair(ma,u));
210 }
211 int tree_DC(int u,int sz){
212 int center=tree_centroid(u,-1,sz).second;
213 int ans=cal(center,0);
214 vis[center]=1;
215 for(size_t i=0;i<g[center].size();++i){
216 int v=g[center][i].first,w=g[center][i].
217 second;
218 if(vis[v])continue;
219 ans-=cal(v,w);
220 ans+=tree_DC(v,size[v]);
221 }
222 return ans;
223 }
224 int main(){
225 while(scanf("%d%d",&n,&k),n||k){
226 init();
227 for(int i=1;i<=n;++i){
228 int u,v,w;
229 scanf("%d%d%d",&u,&v,&w);
230 g[u].push_back(make_pair(v,w));
231 g[v].push_back(make_pair(u,w));
232 }
233 printf("%d\n",tree_DC(1,n));
234 }
235 }

```

10.4 POJ_tree.cpp

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define MAXN 10005
4 int n,k;
5 vector<pair<int,int>> >g[MAXN];
6 int size[MAXN];

```

ACM ICPC Team Reference - NTHU Jinkela

Contents

1 Computational_Geometry	1	2.2 Dynamic_KD_tree.cpp	4	5.6 graphISO.cpp	9	7.14NTT.cpp	15
1.1 formula.tex	1	2.3 kd_tree_replace_segment_tree.cpp	5	5.7 KM.cpp	9	7.15random.cpp	15
1.2 Geometry.cpp	1	2.4 persistent_segment_tree.cpp	5	5.8 MaximumClique.cpp	9	7.16外星模運算.cpp	15
1.3 SmallestCircle.cpp	3	2.5 reference_point.cpp	5	5.9 Minimum_General_Weighted_Matching.cpp	9	7.17模運算模板.cpp	15
1.4 最近點對.cpp	3	2.6 skew_heap.cpp	6	5.10Rectilinear_Steiner_tree.cpp	9	8 String	15
1.5 浮點數誤差模板.cpp	3	2.7 split_merge.cpp	6	5.11treeISO.cpp	10	8.1 AC 自動機.cpp	15
2 Data_Structure	3	2.8 treap.cpp	6	5.12一般圖最大權匹配.cpp	10	8.2 hash.cpp	16
2.1 DLX.cpp	3	2.9 操作分治.cpp	6	6 language	11	8.3 KMP.cpp	16
		2.10 整體二分.cpp	6	6.1 CNF.cpp	11	8.4 manacher.cpp	16
		3 default	7	6.2 earley.cpp	11	8.5 minimal_string_rotation.cpp	16
		3.1 debug.cpp	7	7 Number_Theory	12	8.6 suffix_array_lcp.cpp	16
		3.2 IncStack.cpp	7	7.1 basic.cpp	12	8.7 Z.cpp	17
		3.3 input.cpp	7	7.2 bit_set.cpp	13	9 Tarjan	17
		4 Flow	7	7.3 cantor_expansion.cpp	13	9.1 dominator_tree.cpp	17
		4.1 dinic.cpp	7	7.4 Chinese_remainder_theorem.cpp	13	9.2 tnfsb017_2_sat.cpp	17
		4.2 ISAP.cpp	7	7.5 enumerate.cpp	13	9.3 橋連通分量.cpp	17
		4.3 MinCostMaxFlow.cpp	7	7.6 eulerphi.cpp	13	9.4 雙連通分量 & 割點.cpp	17
		5 Graph	8	7.7 Factor.cpp	13	10 Tree_problem	18
		5.1 Arborescence_EV.cpp	8	7.8 FFT.cpp	14	10.1HeavyLight.cpp	18
		5.2 Augmenting_Path.cpp	8	7.9 find_real_root.cpp	14	10.2LCA.cpp	18
		5.3 Augmenting_Path_multiple.cpp	8	7.10formula.tex	14	10.3link_cut_tree.cpp	18
		5.4 blossom_matching.cpp	8	7.11Gauss_Elimination.cpp	14	10.4POJ_tree.cpp	19
		5.5 formula.tex	8	7.12LinearCongruence.cpp	14		
				7.13Lucas.cpp	15		