

1 Computational Geometry

1.1 SmallestCircle

```

1 using PT=point<T>; using CPT=const PT;
2 PT circumcenter(CPT &a,CPT &b,CPT &c){
3     PT u=b-a, v=c-a;
4     T c1=u.abs2()/2, c2=v.abs2()/2;
5     T d=u.cross(v);
6     return PT(a.x+(v.y*c1-u.y*c2)/d, a.y+(u.x*
7         c2-v.x*c1)/d);
8 }
9 void solve(PT p[], int n, PT &c, T &r2){
10     random_shuffle(p, p+n);
11     c=p[0]; r2=0; // c, r2 = 圓心, 半徑平方
12     for(int i=1; i<n; i++){
13         if((p[i]-c).abs2()>r2){
14             c=p[i]; r2=0;
15         }
16         for(int j=0; j<i; j++){
17             if((p[j]-c).abs2()>r2){
18                 c.x=(p[i].x+p[j].x)/2;
19                 c.y=(p[i].y+p[j].y)/2;
20                 r2=(p[j]-c).abs2();
21             }
22         }
23     }
24 }

```

1.2 Geometry

```

1 const double PI=atan2(0.0, -1.0);
2 template<typename T>
3 struct point{
4     T x,y;
5     point(){}
6     point(const T&x, const T&y):x(x),y(y){}
7     point operator+(const point &b)const{
8         return point(x+b.x, y+b.y); }
9     point operator-(const point &b)const{
10        return point(x-b.x, y-b.y); }
11     point operator*(const T &b)const{
12        return point(x*b, y*b); }
13     point operator/(const T &b)const{
14        return point(x/b, y/b); }
15     bool operator==(const point &b)const{
16        return x==b.x && y==b.y; }
17     T dot(const point &b)const{
18        return x*b.x + y*b.y; }
19     T cross(const point &b)const{
20        return x*b.y - y*b.x; }
21     point normal()const{//求法向量
22        return point(-y, x); }
23     T abs2()const{//向量長度的平方
24        return dot(*this); }
25     T rad(const point &b)const{//兩向量的弧度
26        return fabs(atan2(fabs(cross(b)), dot(b))); }
27     T getA()const{//對x軸的弧度
28        T A=atan2(y, x); //超過180度會變負的

```

```

29     if(A<=-PI/2)A+=PI*2;
30     return A;
31 }
32 };
33 template<typename T>
34 struct line{
35     line(){}
36     point<T> p1,p2;
37     T a,b,c; //ax+by+c=0
38     line(const point<T>&x, const point<T>&y):p1
39         (x),p2(y){}
40     void pton()const{//轉成一般式
41         a=p1.y-p2.y;
42         b=p2.x-p1.x;
43         c=-a*p1.x-b*p1.y;
44     }
45     T ori(const point<T> &p)const{//點和有向直
46         線的關係, >0左邊, =0在線上, <0右邊
47         return (p2-p1).cross(p-p1);
48     }
49     T btw(const point<T> &p)const{//點投影落在
50         線段上<=0
51         return (p1-p).dot(p2-p);
52     }
53     bool point_on_segment(const point<T>&p)
54         const{//點是否在線段上
55         return ori(p)==0 && btw(p)<=0;
56     }
57     T dis2(const point<T> &p, bool is_segment
58         =0)const{//點跟直線/線段的距離平方
59         point<T> v=p2-p1, v1=p-p1;
60         if(is_segment){
61             point<T> v2=p-p2;
62             if(v.dot(v1)<=0) return v1.abs2();
63             if(v.dot(v2)>=0) return v2.abs2();
64         }
65         T tmp=v.cross(v1);
66         return tmp*tmp/v.abs2();
67     }
68     T seg_dis2(const line<T> &l)const{//兩線段
69         距離平方
70         return min({dis2(l.p1, l), dis2(l.p2, l),
71             dis2(p1, l), l.dis2(p2, l)});
72     }
73     point<T> projection(const point<T> &p)
74         const{//點對直線的投影
75         point<T> n=(p2-p1).normal();
76         return p-n*(p-p1).dot(n)/n.abs2();
77     }
78     point<T> mirror(const point<T> &p)const{
79         //點對直線的鏡射, 要先呼叫pton轉成一般式
80         point<T> R;
81         T d=a*a+b*b;
82         R.x=(b*b*p.x-a*a*p.x-2*a*b*p.y-2*a*c)/d;
83         R.y=(a*a*p.y-b*b*p.y-2*a*b*p.x-2*b*c)/d;
84         return R;
85     }
86     bool equal(const line &l)const{//直線相等
87         return ori(l.p1)==0 && ori(l.p2)==0;
88     }
89     bool parallel(const line &l)const{
90         return (p1-p2).cross(l.p1-l.p2)==0;
91     }
92     bool cross_seg(const line &l)const{

```

```

93         return (p2-p1).cross(l.p1-p1)*(p2-p1).
94             cross(l.p2-p1)<=0; //直線是否交線段
95     }
96     int line_intersect(const line &l)const{//
97         直線相交情況, -1無限多點, 1交於一點, 0
98         不相交
99         return parallel(l)?(ori(l.p1)==0?-1:0)
100             :1;
101     }
102     int seg_intersect(const line &l)const{
103         T c1=ori(l.p1), c2=ori(l.p2);
104         T c3=l.ori(p1), c4=l.ori(p2);
105         if(c1==0 && c2==0){ //共線
106             bool b1=btw(l.p1)>=0, b2=btw(l.p2)>=0;
107             T a3=l.btw(p1), a4=l.btw(p2);
108             if(b1 && b2 && a3==0 && a4==0) return 2;
109             if(b1 && b2 && a3>0 && a4==0) return 3;
110             if(b1 && b2 && a3>0 && a4>0) return 0;
111             return -1; //無限交點
112         } else if(c1*c2<=0 && c3*c4<=0) return 1;
113         return 0; //不相交
114     }
115     point<T> line_intersection(const line &l)
116         const{//直線交點*
117         point<T> a=p2-p1, b=l.p2-l.p1, s=l.p1-p1;
118         //if(a.cross(b)==0) return INF;
119         return p1+a*(s.cross(b)/a.cross(b));
120     }
121     point<T> seg_intersection(const line &l)
122         const{//線段交點
123         int res=seg_intersect(l);
124         if(res<=0) assert(0);
125         if(res==2) return p1;
126         if(res==3) return p2;
127         return line_intersection(l);
128     }
129     };
130     template<typename T>
131     struct polygon{
132         polygon(){}
133         vector<point<T> > p; //逆時針順序
134         T area()const{//面積
135             T ans=0;
136             for(int i=p.size()-1, j=0; j<(int)p.size();
137                 i=j++){
138                 ans+=p[i].cross(p[j]);
139             }
140             return ans/2;
141         }
142         point<T> center_of_mass()const{//重心
143             T cx=0, cy=0, w=0;
144             for(int i=p.size()-1, j=0; j<(int)p.size();
145                 i=j++){
146                 T a=p[i].cross(p[j]);
147                 cx+=(p[i].x+p[j].x)*a;
148                 cy+=(p[i].y+p[j].y)*a;
149                 w+=a;
150             }
151             return point<T>(cx/3/w, cy/3/w);
152         }
153     };
154     char ahas(const point<T>& t)const{//點是否
155         在簡單多邊形內, 是的話回傳1, 在邊上回
156         傳-1, 否則回傳0
157     }
158     bool c=0;

```

```

159     for(int i=0, j=p.size()-1; i<p.size(); i=j+1)
160         ++j;
161         if(line<T>(p[i], p[j]).point_on_segment
162             (t)) return -1;
163         else if((p[i].y>t.y) != (p[j].y>t.y) &&
164             t.x<(p[j].x-p[i].x)*(t.y-p[i].y)/(p[j]
165                 .y-p[i].y)+p[i].x)
166             c=!c;
167         return c;
168     }
169     char point_in_convex(const point<T>&x)
170         const{
171         int l=1, r=(int)p.size()-2;
172         while(l<=r){ //點是否在凸多邊形內, 是的話
173             回傳1, 在邊上回傳-1, 否則回傳0
174             int mid=(l+r)/2;
175             T a1=(p[mid]-p[0]).cross(x-p[0]);
176             T a2=(p[mid+1]-p[0]).cross(x-p[0]);
177             if(a1>=0 && a2<=0){
178                 T res=(p[mid+1]-p[mid]).cross(x-p[
179                     mid]);
180                 return res>0?1:(res>0?-1:0);
181             } else if(a1<0) r=mid-1;
182             else l=mid+1;
183         }
184         return 0;
185     }
186     vector<T> getA()const{//凸包邊對x軸的夾角
187     vector<T> res; //一定是遞增的
188     for(size_t i=0; i<p.size(); ++i)
189         res.push_back((p[(i+1)%p.size()]-p[i])
190             .getA());
191     return res;
192     }
193     bool line_intersect(const vector<T>&A,
194         const line<T> &l)const{//0(LogN)
195     int f1=upper_bound(A.begin(), A.end(), (l.
196         p1-l.p2).getA())-A.begin();
197     int f2=upper_bound(A.begin(), A.end(), (l.
198         p2-l.p1).getA())-A.begin();
199     return l.cross_seg(line<T>(p[f1], p[f2]));
200     }
201     polygon cut(const line<T> &l)const{//凸包
202         對直線切割, 得到直線L左側的凸包
203     polygon ans;
204     for(int n=p.size(), i=n-1, j=0; j<n; i=j++){
205         if(l.ori(p[i])>=0){
206             ans.p.push_back(p[i]);
207             if(l.ori(p[j])<0)
208                 ans.p.push_back(l.
209                     line_intersection(line<T>(p[i]
210                         , p[j])));
211         } else if(l.ori(p[j])>0)
212             ans.p.push_back(l.line_intersection(
213                 line<T>(p[i], p[j])));
214         }
215     }
216     return ans;
217     }
218     static bool monotone_chain_cmp(const point
219         <T>& a, const point<T>& b){ //凸包排序函
220         數
221         return (a.x<b.x) || (a.x==b.x && a.y<b.y);
222     }
223 }

```

```

185 void monotone_chain(vector<point<T> > &s){
186     //凸包
187     sort(s.begin(),s.end(),
188         monotone_chain_cmp);
189     p.resize(s.size()+1);
190     int m=0;
191     for(size_t i=0;i<s.size();++i){
192         while(m>=2&&(p[m-1]-p[m-2]).cross(s[i]
193             -p[m-2])<=0)--m;
194         p[m++]=s[i];
195     }
196     for(int i=s.size()-2,t=m+1;i>=0;--i){
197         while(m>=t&&(p[m-1]-p[m-2]).cross(s[i]
198             -p[m-2])<=0)--m;
199         p[m++]=s[i];
200     }
201     if(s.size()>1)--m;
202     p.resize(m);
203 }
204 T diam(){//直徑
205     int n=p.size(),t=1;
206     T ans=0;p.push_back(p[0]);
207     for(int i=0;i<n;i++){
208         point<T> now=p[i+1]-p[i];
209         while(now.cross(p[t+1]-p[i])>now.cross
210             (p[t]-p[i]))t=(t+1)%n;
211         ans=max(ans,(p[i]-p[t]).abs2());
212     }
213     return p.pop_back(),ans;
214 }
215 T min_cover_rectangle(){//最小覆蓋矩形
216     int n=p.size(),t=1,r=1,l=1;
217     if(n<3)return 0;//也可以做最小周長矩形
218     T ans=1e99;p.push_back(p[0]);
219     for(int i=0;i<n;i++){
220         point<T> now=p[i+1]-p[i];
221         while(now.cross(p[t+1]-p[i])>now.cross
222             (p[t]-p[i]))t=(t+1)%n;
223         while(now.dot(p[r+1]-p[i])>now.dot(p[r]
224             -p[i]))r=(r+1)%n;
225         if(!l)l=r;
226         while(now.dot(p[l+1]-p[i])<=now.dot(p[
227             l]-p[i]))l=(l+1)%n;
228         T d=now.abs2();
229         T tmp=now.cross(p[t]-p[i])*(now.dot(p[
230             r]-p[i])-now.dot(p[l]-p[i]))/d;
231         ans=min(ans,tmp);
232     }
233     return p.pop_back(),ans;
234 }
235 T dis2(polygon &p1){//凸包最近距離平方
236     vector<point<T> > &P=p,Q=p1.p;
237     int n=P.size(),m=Q.size(),l=0,r=0;
238     for(int i=0;i<n;++i)if(P[i].y<P[l].y)l=i;
239     for(int i=0;i<m;++i)if(Q[i].y<Q[r].y)r=i;
240     P.push_back(P[0]),Q.push_back(Q[0]);
241     T ans=1e99;
242     for(int i=0;i<n;++i){
243         while((P[l]-P[l+1]).cross(Q[r+1]-Q[r])
244             <0)r=(r+1)%m;
245         ans=min(ans,line<T>(P[l],P[l+1]).
246             seg_dis2(line<T>(Q[r],Q[r+1])));
247         l=(l+1)%n;
248     }
249     return P.pop_back(),Q.pop_back(),ans;
250 }
251 static char sign(const point<T>&t){
252     return (t.y==0?t.x:t.y)<0;
253 }
254 static bool angle_cmp(const line<T>& A,
255     const line<T>& B){
256     point<T> a=A.p2-A.p1,b=B.p2-B.p1;
257     return sign(a)<sign(b)||((sign(a)==sign(b)
258         )&&a.cross(b)>0);
259 }
260 int halfplane_intersection(vector<line<T>
261     > &s){//半平面交
262     sort(s.begin(),s.end(),angle_cmp);//線段
263     //左側為該線段半平面
264     int L,R,n=s.size();
265     vector<point<T> > px(n);
266     vector<line<T> > q(n);
267     q[L=R=0]=s[0];
268     for(int i=1;i<n;++i){
269         while(L<R&&s[i].ori(px[R-1])<=0)--R;
270         while(L<R&&s[i].ori(px[L])<=0)++L;
271         q[++R]=s[i];
272         if(q[R].parallel(q[R-1])){
273             --R;
274             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
275         }
276         if(L<R)px[R-1]=q[R-1].
277             line_intersection(q[R]);
278     }
279     while(L<R&&q[L].ori(px[R-1])<=0)--R;
280     p.clear();
281     if(R-L<=1)return 0;
282     px[R]=q[R].line_intersection(q[L]);
283     for(int i=L;i<R;++i)p.push_back(px[i]);
284     return R-L+1;
285 }
286 }
287 template<typename T>
288 struct triangle{
289     point<T> a,b,c;
290     triangle(const point<T> &a,const point<T>
291         &b,const point<T> &c):a(a),b(b),c(c){}
292     T area()const{
293         T t=(b-a).cross(c-a)/2;
294         return t>0?t:-t;
295     }
296     point<T> barycenter()const{//重心
297         return (a+b+c)/3;
298     }
299     point<T> circumcenter()const{//外心
300         static line<T> u,v;
301         u.p1=(a+b)/2;
302         u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-
303             b.x);
304         v.p1=(a+c)/2;
305         v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-
306             c.x);
307         return u.line_intersection(v);
308     }
309     point<T> incenter()const{//內心
310         T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2
311             ()),C=sqrt((a-b).abs2());
312         return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+
313             B*b.y+C*c.y)/(A+B+C);
314     }
315 }
316 point<T> perpcenter()const{//垂心
317     return barycenter()*3-circumcenter()*2;
318 }
319 }
320 template<typename T>
321 struct point3D{
322     T x,y,z;
323     point3D(){}
324     point3D(const T&x,const T&y,const T&z):x(x
325         ),y(y),z(z){}
326     point3D operator+(const point3D &b)const{
327         return point3D(x+b.x,y+b.y,z+b.z);
328     }
329     point3D operator-(const point3D &b)const{
330         return point3D(x-b.x,y-b.y,z-b.z);
331     }
332     point3D operator*(const T &b)const{
333         return point3D(x*b,y*b,z*b);
334     }
335     point3D operator/(const T &b)const{
336         return point3D(x/b,y/b,z/b);
337     }
338     bool operator==(const point3D &b)const{
339         return x==b.x&&y==b.y&&z==b.z;
340     }
341     T dot(const point3D &b)const{
342         return x*b.x+y*b.y+z*b.z;
343     }
344     point3D cross(const point3D &b)const{
345         return point3D(y*b.z-z*b.y,z*b.x-x*b.z,x
346             *b.y-y*b.x);
347     }
348     T abs2()const{//向量長度的平方
349         return dot(*this);
350     }
351     T area2(const point3D &b)const{//和b、原點
352         //圍成面積的平方
353         return cross(b).abs2()/4;
354     }
355 }
356 template<typename T>
357 struct line3D{
358     point3D<T> p1,p2;
359     line3D(){}
360     line3D(const point3D<T> &p1,const point3D<
361         T> &p2):p1(p1),p2(p2){}
362     T dis2(const point3D<T> &p,bool is_segment
363         =0)const{//點跟直線/線段的距離平方
364         point3D<T> v=p2-p1,v1=p-p1;
365         if(is_segment){
366             point3D<T> v2=p-p2;
367             if(v.dot(v1)<=0)return v1.abs2();
368             if(v.dot(v2)>=0)return v2.abs2();
369         }
370         point3D<T> tmp=v.cross(v1);
371         return tmp.abs2()/v.abs2();
372     }
373     pair<point3D<T>,point3D<T> > closest_pair(
374         const line3D<T> &l)const{
375         point3D<T> v1=(p1-p2),v2=(l.p1-l.p2);
376         point3D<T> N=v1.cross(v2),ab(p1-l.p1);
377         //if(N.abs2()==0)return NULL; 平行或重合
378         T tmp=N.dot(ab),ans=tmp*tmp/N.abs2();//
379         //最近點對距離
380         point3D<T> d1=p2-p1,d2=l.p2-l.p1,D=d1.
381             cross(d2),G=l.p1-p1;
382         T t1=(G.cross(d2)).dot(D)/D.abs2();
383         T t2=(G.cross(d1)).dot(D)/D.abs2();
384         return make_pair(p1+d1*t1,l.p1+d2*t2);
385     }
386     bool same_side(const point3D<T> &a,const
387         point3D<T> &b)const{
388         return (p2-p1).cross(a-p1).dot((p2-p1).
389             cross(b-p1))>0;
390     }
391 };
392 template<typename T>
393 struct plane{
394     point3D<T> p0,n;//平面上的點和法向量
395     plane(){}
396     plane(const point3D<T> &p0,const point3D<T>
397         &n):p0(p0),n(n){}
398     T dis2(const point3D<T> &p)const{//點到平
399         //面距離的平方
400         T tmp=(p-p0).dot(n);
401         return tmp*tmp/n.abs2();
402     }
403     point3D<T> projection(const point3D<T> &p)
404         const{
405         return p-n*(p-p0).dot(n)/n.abs2();
406     }
407     point3D<T> line_intersection(const line3D<
408         T> &l)const{
409         T tmp=n.dot(l.p2-l.p1);//等於0表示平行或
410             重合該平面
411         return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/
412             tmp);
413     }
414     line3D<T> plane_intersection(const plane &
415         p1)const{
416         point3D<T> e=n.cross(p1.n),v=n.cross(e);
417         T tmp=p1.n.dot(v);//等於0表示平行或重合
418             該平面
419         point3D<T> q=p0+(v*(p1.n.dot(p1.p0-p0))/
420             tmp);
421         return line3D<T>(q,q+e);
422     }
423 };
424 template<typename T>
425 struct triangle3D{
426     point3D<T> a,b,c;
427     triangle3D(){}
428     triangle3D(const point3D<T> &a,const
429         point3D<T> &b,const point3D<T> &c):a(a
430             ),b(b),c(c){}
431     bool point_in(const point3D<T> &p)const{//
432         //點在該平面上的投影在三角形中
433         return line3D<T>(b,c).same_side(p,a)&&
434             line3D<T>(a,c).same_side(p,b)&&
435             line3D<T>(a,b).same_side(p,c);
436     }
437 };
438 template<typename T>
439 struct tetrahedron{//四面體
440     point3D<T> a,b,c,d;
441     tetrahedron(){}
442     tetrahedron(const point3D<T> &a,const
443         point3D<T> &b,const point3D<T> &c,
444         const point3D<T> &d):a(a),b(b),c(c),d(
445             d){}
446     T volume6()const{//體積的六倍
447         return (d-a).dot((b-a).cross(c-a));
448     }
449     point3D<T> centroid()const{
450         return (a+b+c+d)/4;
451     }
452 };

```

```

395 bool point_in(const point3D<T> &p) const {
396     return triangle3D<T>(a,b,c).point_in(p)
        && triangle3D<T>(c,d,a).point_in(p);
397 }
398 };
399 template<typename T>
400 struct convexhull3D {
401     static const int MAXN=1005;
402     struct face {
403         int a,b,c;
404         face(int a,int b,int c):a(a),b(b),c(c){}
405     };
406     vector<point3D<T>> pt;
407     vector<face> ans;
408     int fid[MAXN][MAXN];
409     void build() {
410         int n=pt.size();
411         ans.clear();
412         memset(fid,0,sizeof(fid));
413         ans.emplace_back(0,1,2); // 注意不能共線
414         ans.emplace_back(2,1,0);
415         int ftop = 0;
416         for(int i=3, ftop=1; i<n; ++i, ++ftop) {
417             vector<face> next;
418             for(auto &f:ans) {
419                 T d=(pt[i]-pt[f.a]).dot((pt[f.b]-pt[f.a]).cross(pt[f.c]-pt[f.a]));
420                 if(d<=0) next.push_back(f);
421                 int ff=0;
422                 if(d>0) ff=ftop;
423                 else if(d<0) ff=-ftop;
424                 fid[f.a][f.b]=fid[f.b][f.c]=fid[f.c][f.a]=ff;
425             }
426             for(auto &f:ans) {
427                 if(fid[f.a][f.b]>0 && fid[f.a][f.b]
                    !=fid[f.b][f.a])
428                     next.emplace_back(f.a,f.b,i);
429                 if(fid[f.b][f.c]>0 && fid[f.b][f.c]
                    !=fid[f.c][f.b])
430                     next.emplace_back(f.b,f.c,i);
431                 if(fid[f.c][f.a]>0 && fid[f.c][f.a]
                    !=fid[f.a][f.c])
432                     next.emplace_back(f.c,f.a,i);
433             }
434             ans=next;
435         }
436     }
437     point3D<T> centroid() const {
438         point3D<T> res(0,0,0);
439         T vol=0;
440         for(auto &f:ans) {
441             T tmp=pt[f.a].dot(pt[f.b].cross(pt[f.c]
                ));
442             res=res+(pt[f.a]+pt[f.b]+pt[f.c])*tmp;
443             vol+=tmp;
444         }
445         return res/(vol*4);
446     }
447 };

```

1.3 最近點對

```

1 template<typename _IT=point<T>* >
2 T closest_pair(_IT L, _IT R) {
3     if(R-L <= 1) return INF;
4     _IT mid = L+(R-L)/2;
5     T x = mid->x;
6     T d = min(closest_pair(L,mid),closest_pair(
            mid,R));
7     inplace_merge(L, mid, R, ycmp);
8     static vector<point> b; b.clear();
9     for(auto u=L; u<R; ++u) {
10         if((u->x-x)*(u->x-x)>=d) continue;
11         for(auto v=b.rbegin(); v!=b.rend(); ++v) {
12             T dx=u->x-v->x, dy=u->y-v->y;
13             if(dy*dy>=d) break;
14             d=min(d,dx*dx+dy*dy);
15         }
16         b.push_back(*u);
17     }
18     return d;
19 }
20 T closest_pair(vector<point<T>> &v) {
21     sort(v.begin(),v.end(),xcmp);
22     return closest_pair(v.begin(),v.end());
23 }

```

2 Data Structure

2.1 文藝平衡樹

```

1 struct Node {
2     Node* ch[2];
3     int val, prio;
4     int cnt;
5     int siz;
6     bool to_rev = false; // 需要把这个子树下
        的每一个节点都翻转过来
7
8     Node(int _val) : val(_val), cnt(1), siz(1)
        {
9         ch[0] = ch[1] = nullptr;
10        prio = rand();
11    }
12
13    int upd_siz() {
14        siz = cnt;
15        if (ch[0] != nullptr) siz += ch[0]->siz;
16        if (ch[1] != nullptr) siz += ch[1]->siz;
17        return siz;
18    }
19
20    void pushdown() {
21        swap(ch[0], ch[1]);
22        if (ch[0] != nullptr) ch[0]->to_rev ^=
            1;
23        // 如果原来子节点也要翻转，那两次翻转就
            抵消了，如果子节点不翻转，那这个
24        // tag 就需要继续被 push 到子节点上
25        if (ch[1] != nullptr) ch[1]->to_rev ^=
            1;
26        to_rev = false;

```

```

27    }
28
29    void check_tag() {
30        if (to_rev) pushdown();
31    }
32
33    };
34    struct Seg_treap {
35        Node* root;
36        #define siz(_) (_ == nullptr ? 0 : _->siz)
37
38        pair<Node*, Node*> split(Node* cur, int sz
            ) {
39            // 按照树的大小划分
40            if (cur == nullptr) return {nullptr,
                nullptr};
41            cur->check_tag();
42            if (siz <= siz(cur->ch[0])) {
43                // 左边的子树就够了
44                auto temp = split(cur->ch[0], sz);
45                // 左边的子树不一定全部需要 temp.
                second 是不需要的
46                cur->ch[0] = temp.second;
47                cur->upd_siz();
48                return {temp.first, cur};
49            } else {
50                // 左边的加上右边的一部分 (当然也包括
                这个节点本身)
51                auto temp = split(cur->ch[1], sz - siz
                    (cur->ch[0]) - 1);
52                cur->ch[1] = temp.first;
53                cur->upd_siz();
54                return {cur, temp.second};
55            }
56        }
57
58        Node* merge(Node* sm, Node* bg) {
59            // small, big
60            if (sm == nullptr && bg == nullptr)
                return nullptr;
61            if (sm != nullptr && bg == nullptr)
                return sm;
62            if (sm == nullptr && bg != nullptr)
                return bg;
63            sm->check_tag(), bg->check_tag();
64            if (sm->prio < bg->prio) {
65                sm->ch[1] = merge(sm->ch[1], bg);
66                sm->upd_siz();
67                return sm;
68            } else {
69                bg->ch[0] = merge(sm, bg->ch[0]);
70                bg->upd_siz();
71                return bg;
72            }
73        }
74
75        void insert(int val) {
76            auto temp = split(root, val);
77            auto l_tr = split(temp.first, val - 1);
78            Node* new_node;
79            if (l_tr.second == nullptr) new_node =
                new Node(val);
80            Node* l_tr_combined =
                merge(l_tr.first, l_tr.second ==
                    nullptr ? new_node : l_tr.second

```

```

81            );
82            root = merge(l_tr_combined, temp.second);
83        };
84
85        void seg_rev(int l, int r) {
86            // 这里的 less 和 more 是相对于 l 的
87            auto less = split(root, l - 1);
88            // 所有小于等于 l - 1 的会在 less 的左边
89            auto more = split(less.second, r - l +
                1);
90            // 拿出从 l 开始的前 r - l + 1 个
91            more.first->to_rev = true;
92            root = merge(less.first, merge(more.
                first, more.second));
93        }
94
95        void print(Node* cur) {
96            if (cur == nullptr) return;
97            cur->check_tag();
98            print(cur->ch[0]);
99            cout << cur->val << " ";
100            print(cur->ch[1]);
101        }
102    };
103
104    Seg_treap tr;
105
106    int main() {
107        srand(time(0));
108        int n, m;
109        cin >> n >> m;
110        for (int i = 1; i <= n; i++) tr.insert(i);
111        while (m--) {
112            int l, r;
113            cin >> l >> r;
114            tr.seg_rev(l, r);
115        }
116        tr.print(tr.root);
117    }

```

2.2 DisjointSet

```

1 struct DisjointSetUnion {
2     public:
3     DisjointSetUnion()
4         : _n(0) {}
5     explicit DisjointSetUnion(int n)
6         : _n(n)
7         , parent_or_size(n, -1) {}
8
9     int merge(int a, int b) {
10        assert(0 <= a && a < _n);
11        assert(0 <= b && b < _n);
12        int x = leader(a), y = leader(b);
13        if (x == y) return x;
14        if (-parent_or_size[x] < -
            parent_or_size[y])
15            std::swap(x, y);
16        parent_or_size[x] += parent_or_size[y];
17        parent_or_size[y] = x;

```

```

18     return x;
19 }
20
21 bool same(int a, int b) {
22     assert(0 <= a && a < _n);
23     assert(0 <= b && b < _n);
24     return leader(a) == leader(b);
25 }
26
27 int leader(int a) {
28     assert(0 <= a && a < _n);
29     if (parent_or_size[a] < 0) return a;
30     return parent_or_size[a] = leader(
31         parent_or_size[a]);
32 }
33
34 int size(int a) {
35     assert(0 <= a && a < _n);
36     return -parent_or_size[leader(a)];
37 }
38
39 std::vector<std::vector<int>> groups() {
40     std::vector<int> leader_buf(_n),
41         group_size(_n);
42     for (int i = 0; i < _n; i++) {
43         leader_buf[i] = leader(i);
44         group_size[leader_buf[i]]++;
45     }
46     std::vector<std::vector<int>> result(
47         _n);
48     for (int i = 0; i < _n; i++) {
49         result[i].reserve(group_size[i]);
50     }
51     for (int i = 0; i < _n; i++) {
52         result[leader_buf[i]].push_back(i);
53     }
54     result.erase(
55         std::remove_if(
56             result.begin(), result.end(),
57             [&](const std::vector<int>& v) {
58                 return v.empty();
59             }),
60         result.end());
61     return result;
62 }
63
64 private:
65 int _n;
66 // root node: -1 * component size
67 // otherwise: parent
68 std::vector<int> parent_or_size;
69 };

```

2.3 treap

```

1 struct Node {
2     Node *ch[2];
3     int val, prio;
4     int cnt;
5     int siz;
6
7     Node(int _val) : val(_val), cnt(1), siz(1)
8     {

```

```

9         ch[0] = ch[1] = nullptr;
10        prio = rand();
11    }
12
13    Node(Node *_node) {
14        val = _node->val, prio = _node->prio,
15        cnt = _node->cnt, siz = _node->siz;
16    }
17
18    void upd_siz() {
19        siz = cnt;
20        if (ch[0] != nullptr) siz += ch[0]->siz;
21        if (ch[1] != nullptr) siz += ch[1]->siz;
22    }
23
24    struct none_rot_treap {
25        #define _3 second.second
26        #define _2 second.first
27        Node *root;
28
29        pair<Node *, Node *> split(Node *cur, int
30            key) {
31            if (cur == nullptr) return {nullptr,
32                nullptr};
33            if (cur->val <= key) {
34                auto temp = split(cur->ch[1], key);
35                cur->ch[1] = temp.first;
36                cur->upd_siz();
37                return {cur, temp.second};
38            } else {
39                auto temp = split(cur->ch[0], key);
40                cur->ch[0] = temp.second;
41                cur->upd_siz();
42                return {temp.first, cur};
43            }
44        }
45
46        tuple<Node *, Node *, Node *> split_by_rk(
47            Node *cur, int rk) {
48            if (cur == nullptr) return {nullptr,
49                nullptr, nullptr};
50            int ls_siz = cur->ch[0] == nullptr ? 0 :
51                cur->ch[0]->siz;
52            if (rk <= ls_siz) {
53                Node *l, *mid, *r;
54                tie(l, mid, r) = split_by_rk(cur->ch
55                    [0], rk);
56                cur->ch[0] = r;
57                cur->upd_siz();
58                return {l, mid, cur};
59            } else if (rk <= ls_siz + cur->cnt) {
60                Node *lt = cur->ch[0];
61                Node *rt = cur->ch[1];
62                cur->ch[0] = cur->ch[1] = nullptr;
63                return {lt, cur, rt};
64            } else {
65                Node *l, *mid, *r;
66                tie(l, mid, r) = split_by_rk(cur->ch
67                    [1], rk - ls_siz - cur->cnt);
68                cur->ch[1] = l;
69                cur->upd_siz();
70                return {cur, mid, r};
71            }
72        }
73    };

```

```

74    Node *merge(Node *u, Node *v) {
75        if (u == nullptr && v == nullptr) return
76            nullptr;
77        if (u != nullptr && v == nullptr) return
78            u;
79        if (v != nullptr && u == nullptr) return
80            v;
81        if (u->prio < v->prio) {
82            u->ch[1] = merge(u->ch[1], v);
83            u->upd_siz();
84            return u;
85        } else {
86            v->ch[0] = merge(u, v->ch[0]);
87            v->upd_siz();
88            return v;
89        }
90    }
91
92    void insert(int val) {
93        auto temp = split(root, val);
94        auto l_tr = split(temp.first, val - 1);
95        Node *new_node;
96        if (l_tr.second == nullptr) {
97            new_node = new Node(val);
98        } else {
99            l_tr.second->cnt++;
100            l_tr.second->upd_siz();
101        }
102        Node *l_tr_combined =
103            merge(l_tr.first, l_tr.second ==
104                nullptr ? new_node : l_tr.second
105                );
106        root = merge(l_tr_combined, temp.second);
107    }
108
109    void del(int val) {
110        auto temp = split(root, val);
111        auto l_tr = split(temp.first, val - 1);
112        if (l_tr.second->cnt > 1) {
113            l_tr.second->cnt--;
114            l_tr.second->upd_siz();
115            l_tr.first = merge(l_tr.first, l_tr.
116                second);
117        } else {
118            if (temp.first == l_tr.second) {
119                temp.first = nullptr;
120            }
121            delete l_tr.second;
122            l_tr.second = nullptr;
123        }
124        root = merge(l_tr.first, temp.second);
125    }
126
127    int qrank_by_val(Node *cur, int val) {
128        auto temp = split(cur, val - 1);
129        int ret = (temp.first == nullptr ? 0 :
130            temp.first->siz) + 1;
131        root = merge(temp.first, temp.second);
132        return ret;
133    }
134
135    int qval_by_rank(Node *cur, int rk) {
136        Node *l, *mid, *r;
137        tie(l, mid, r) = split_by_rk(cur, rk);
138        int ret = mid->val;
139    }

```

```

140    root = merge(merge(l, mid), r);
141    return ret;
142 }
143
144 int qprev(int val) {
145     auto temp = split(root, val - 1);
146     int ret = qval_by_rank(temp.first, temp.
147         first->siz);
148     root = merge(temp.first, temp.second);
149     return ret;
150 }
151
152 int qnex(int val) {
153     auto temp = split(root, val);
154     int ret = qval_by_rank(temp.second, 1);
155     root = merge(temp.first, temp.second);
156     return ret;
157 }
158
159 };
160
161 none_rot_treap tr;
162
163 int main() {
164     srand(time(0));
165     int t;
166     scanf("%d", &t);
167     while (t--) {
168         int mode;
169         int num;
170         scanf("%d%d", &mode, &num);
171         switch (mode) {
172             case 1:
173                 tr.insert(num);
174                 break;
175             case 2:
176                 tr.del(num);
177                 break;
178             case 3:
179                 printf("%d\n", tr.qrank_by_val(tr.
180                     root, num));
181                 break;
182             case 4:
183                 printf("%d\n", tr.qval_by_rank(tr.
184                     root, num));
185                 break;
186             case 5:
187                 printf("%d\n", tr.qprev(num));
188                 break;
189             case 6:
190                 printf("%d\n", tr.qnex(num));
191                 break;
192         }
193     }
194 }

```

2.4 fenwick

```

1 namespace internal {
2 template <class T>
3 using to_unsigned_t = typename to_unsigned<T
4     >::type;
5 } // namespace internal

```



```

6 template <class T>
7 struct FenwickTree {
8     using U = internal::to_unsigned_t<T>;
9
10 public:
11     FenwickTree() : _n(0) {}
12     explicit FenwickTree(int n) : _n(n), data(
13         n) {}
14
15     void add(int p, T x) {
16         assert(0 <= p && p < _n);
17         p++;
18         while (p <= _n) {
19             data[p - 1] += U(x);
20             p += p & -p;
21         }
22
23     T sum(int l, int r) {
24         assert(0 <= l && l <= r && r <= _n);
25         return sum(r) - sum(l);
26     }
27
28 private:
29     int _n;
30     std::vector<U> data;
31
32     U sum(int r) {
33         U s = 0;
34         while (r > 0) {
35             s += data[r - 1];
36             r -= r & -r;
37         }
38         return s;
39     }
40 };

```

2.5 segment tree

```

1 template <typename T>
2 class SegmentTree {
3 private:
4     std::vector<T> tree, lazy;
5     std::vector<bool> updated;
6     int n;
7
8     void build(int node, int start, int end,
9         const std::vector<T>& arr) {
10         if (start == end) {
11             tree[node] = arr[start];
12         } else {
13             int mid = (start + end) / 2;
14             build(2 * node, start, mid, arr);
15             build(2 * node + 1, mid + 1, end, arr);
16             tree[node] = tree[2 * node] + tree[2 *
17                 node + 1];
18         }
19
20     void updateRange(int node, int start, int
21         end, int l, int r, T val) {
22         if (updated[node]) {

```

```

23             tree[node] = lazy[node] * (end - start
24                 + 1);
25             if (start != end) {
26                 lazy[node * 2] = lazy[node];
27                 lazy[node * 2 + 1] = lazy[node];
28                 updated[node * 2] = updated[node];
29                 updated[node * 2 + 1] = updated[node];
30             }
31             updated[node] = false;
32         }
33         if (start > end or start > r or end < l)
34             return;
35         if (start >= l and end <= r) {
36             tree[node] = val * (end - start + 1);
37             if (start != end) {
38                 lazy[node * 2] = val;
39                 lazy[node * 2 + 1] = val;
40                 updated[node * 2] = true;
41                 updated[node * 2 + 1] = true;
42             }
43             return;
44         }
45         int mid = (start + end) / 2;
46         updateRange(node * 2, start, mid, l, r,
47             val);
48         updateRange(node * 2 + 1, mid + 1, end,
49             l, r, val);
50         tree[node] = tree[node * 2] + tree[node
51             * 2 + 1];
52
53     T queryRange(int node, int start, int end,
54         int l, int r) {
55         if (start > end or start > r or end < l)
56             return 0;
57         if (updated[node]) {
58             tree[node] = lazy[node] * (end - start
59                 + 1);
60             if (start != end) {
61                 lazy[node * 2] = lazy[node];
62                 lazy[node * 2 + 1] = lazy[node];
63                 updated[node * 2] = updated[node];
64                 updated[node * 2 + 1] = updated[node];
65             }
66             updated[node] = false;
67         }
68         if (start >= l and end <= r) return tree
69             [node];
70         int mid = (start + end) / 2;
71         T p1 = queryRange(node * 2, start, mid,
72             l, r);
73         T p2 = queryRange(node * 2 + 1, mid + 1,
74             end, l, r);
75         return p1 + p2;
76     }
77
78 public:
79     SegmentTree(const std::vector<T>& arr) {
80         n = arr.size();
81         tree.resize(4 * n);
82         lazy.resize(4 * n);
83         updated.resize(4 * n, false);
84         build(1, 0, n - 1, arr);
85     }

```

2.6 undo disjoint set

```

1 struct DisjointSet {
2     // save() is like recursive
3     // undo() is like return
4     int n, fa[MXN], sz[MXN];
5     vector<pair<int*, int*>> h;
6     vector<int> sp;
7     void init(int tn) {
8         n=tn;
9         for (int i=0; i<n; i++) sz[fa[i]=i]=1;
10        sp.clear(); h.clear();
11    }
12    void assign(int *k, int v) {
13        h.PB({k, *k});
14        *k=v;
15    }
16    void save() { sp.PB(SZ(h)); }
17    void undo() {
18        assert(!sp.empty());
19        int last=sp.back(); sp.pop_back();
20        while (SZ(h)!=last) {
21            auto x=h.back(); h.pop_back();
22            *x.F=x.S;
23        }
24    }
25    int f(int x) {
26        while (fa[x]!=x) x=fa[x];
27        return x;
28    }
29    void uni(int x, int y) {
30        x=f(x); y=f(y);
31        if (x==y) return;
32        if (sz[x]<sz[y]) swap(x, y);
33        assign(&sz[x], sz[x]+sz[y]);
34        assign(&fa[y], x);
35    }
36 }djs;

```

3 Flow

3.1 min cost flow

```

1 namespace internal {
2
3 template <class T>
4 struct simple_queue {
5     std::vector<T> payload;
6     int pos = 0;

```

```

7     void reserve(int n) { payload.reserve(n);
8     }
9     int size() const { return int(payload.size()
10         ()) - pos; }
11     bool empty() const { return pos == int(
12         payload.size()); }
13     void push(const T& t) { payload.push_back(
14         t); }
15     T& front() { return payload[pos]; }
16     void clear() {
17         payload.clear();
18         pos = 0;
19     }
20     void pop() { pos++; }
21 };
22
23 template <class E>
24 struct csr {
25     std::vector<int> start;
26     std::vector<E> elist;
27     explicit csr(int n, const std::vector<std
28         ::pair<int, E>>& edges)
29         : start(n + 1), elist(edges.size()) {
30         for (auto e : edges) {
31             start[e.first + 1]++;
32         }
33         for (int i = 1; i <= n; i++) {
34             start[i] += start[i - 1];
35         }
36         auto counter = start;
37         for (auto e : edges) {
38             elist[counter[e.first]++] = e.second;
39         }
40     }
41 };
42
43 // namespace internal
44
45 template <class Cap, class Cost>
46 struct MinCostFlowGraph {
47 public:
48     MinCostFlowGraph() {}
49     explicit MinCostFlowGraph(int n) : _n(n)
50     {}
51
52     int add_edge(int from, int to, Cap cap,
53         Cost cost) {
54         assert(0 <= from && from < _n);
55         assert(0 <= to && to < _n);
56         assert(0 <= cap);
57         assert(0 <= cost);
58         int m = int(_edges.size());
59         _edges.push_back({from, to, cap, 0, cost
60             });
61         return m;
62     }
63
64     struct edge {
65         int from, to;
66         Cap cap, flow;
67         Cost cost;
68     };
69
70     edge get_edge(int i) {
71         int m = int(_edges.size());
72         assert(0 <= i && i < m);

```

```

65     return _edges[i];
66 }
67 std::vector<edge> edges() { return _edges;
68 }
69 std::pair<Cap, Cost> flow(int s, int t) {
70     return flow(s, t, std::numeric_limits<
71         Cap>::max());
72 }
73 std::pair<Cap, Cost> flow(int s, int t,
74     Cap flow_limit) {
75     return slope(s, t, flow_limit).back();
76 }
77 std::vector<std::pair<Cap, Cost>> slope(
78     int s, int t) {
79     return slope(s, t, std::numeric_limits<
80         Cap>::max());
81 }
82 std::vector<std::pair<Cap, Cost>> slope(
83     int s, int t, Cap flow_limit) {
84     assert(0 <= s && s < _n);
85     assert(0 <= t && t < _n);
86     assert(s != t);
87
88     int m = int(_edges.size());
89     std::vector<int> edge_idx(m);
90
91     auto g = [&]() {
92         std::vector<int> degree(_n), redge_idx
93             (m);
94         std::vector<std::pair<int, _edge>>
95             elist;
96         elist.reserve(2 * m);
97         for (int i = 0; i < m; i++) {
98             auto e = _edges[i];
99             edge_idx[i] = degree[e.from]++;
100             redge_idx[i] = degree[e.to]++;
101             elist.push_back({e.from, {e.to, -1,
102                 e.cap - e.flow, e.cost}});
103             elist.push_back({e.to, {e.from, -1,
104                 e.flow, -e.cost}});
105         }
106         auto _g = internal::csr<_edge>(_n,
107             elist);
108         for (int i = 0; i < m; i++) {
109             auto e = _edges[i];
110             edge_idx[i] += _g.start[e.from];
111             redge_idx[i] += _g.start[e.to];
112             _g.elist[edge_idx[i]].rev =
113                 redge_idx[i];
114             _g.elist[redge_idx[i]].rev =
115                 edge_idx[i];
116         }
117         return _g;
118     }();
119
120     auto result = slope(g, s, t, flow_limit)
121         ;
122
123     for (int i = 0; i < m; i++) {
124         auto e = g.elist[edge_idx[i]];
125         _edges[i].flow = _edges[i].cap - e.cap
126             ;
127     }
128
129     return result;
130 }
131
132 private:
133 int _n;
134 std::vector<edge> _edges;
135
136 // inside edge
137 struct _edge {
138     int to, rev;
139     Cap cap;
140     Cost cost;
141 };
142
143 std::vector<std::pair<Cap, Cost>> slope(
144     internal::csr<_edge>& g, int s, int t,
145     Cap flow_limit) {
146     // variants (C = maxcost):
147     // -(n-1)C <= dual[s] <= dual[i] <= dual
148     // [t] = 0
149     // reduced cost (= e.cost + dual[e.from]
150     // - dual[e.to]) >= 0 for all edge
151
152     // dual_dist[i] = (dual[i], dist[i])
153     std::vector<std::pair<Cost, Cost>>
154         dual_dist(_n);
155     std::vector<int> prev_e(_n);
156     std::vector<bool> vis(_n);
157     struct Q {
158         Cost key;
159         int to;
160         bool operator<(Q r) const { return key
161             > r.key; }
162     };
163     std::vector<int> que_min;
164     std::vector<Q> que;
165     auto dual_ref = [&]() {
166         for (int i = 0; i < _n; i++) {
167             dual_dist[i].second = std:::
168                 numeric_limits<Cost>::max();
169         }
170     };
171     std::fill(vis.begin(), vis.end(),
172         false);
173     que_min.clear();
174     que.clear();
175
176     // que[0..heap_r) was heapified
177     size_t heap_r = 0;
178
179     dual_dist[s].second = 0;
180     que_min.push_back(s);
181     while (!que_min.empty() || !que.empty
182         ()) {
183         int v;
184         if (!que_min.empty()) {
185             v = que_min.back();
186             que_min.pop_back();
187         } else {
188             while (heap_r < que.size()) {
189                 heap_r++;
190                 std::push_heap(que.begin(), que.
191                     begin() + heap_r);
192             }
193             v = que.front().to;
194             std::pop_heap(que.begin(), que.end
195                 ());
196             que.pop_back();
197         }
198         heap_r--;
199         if (vis[v]) continue;
200         vis[v] = true;
201         if (v == t) break;
202         // dist[v] = shortest(s, v) + dual[s
203             ] - dual[v]
204         // dist[v] >= 0 (all reduced cost
205             are positive)
206         // dist[v] <= (n-1)C
207         Cost dual_v = dual_dist[v].first,
208             dist_v = dual_dist[v].second;
209         for (int i = g.start[v]; i < g.start
210             [v + 1]; i++) {
211             auto e = g.elist[i];
212             if (!e.cap) continue;
213             // [-dual[e.to] + dual[v]] <= (n
214                 -1)C
215             // cost <= C - -(n-1)C + 0 = nC
216             Cost cost = e.cost - dual_dist[e.
217                 to].first + dual_v;
218             if (dual_dist[e.to].second -
219                 dist_v > cost) {
220                 Cost dist_to = dist_v + cost;
221                 dual_dist[e.to].second = dist_to
222                     ;
223                 prev_e[e.to] = e.rev;
224                 if (dist_to == dist_v) {
225                     que_min.push_back(e.to);
226                 } else {
227                     que.push_back(Q{dist_to, e.to
228                         });
229                 }
230             }
231         }
232     }
233
234     c = std::min(c, g.elist[g.elist[
235         prev_e[v]].rev].cap);
236 }
237
238 for (int v = t; v != s; v = g.elist[
239     prev_e[v]].to) {
240     auto& e = g.elist[prev_e[v]];
241     e.cap += c;
242     g.elist[e.rev].cap -= c;
243 }
244 Cost d = -dual_dist[s].first;
245 flow += c;
246 cost += c * d;
247 if (prev_cost_per_flow == d) {
248     result.pop_back();
249 }
250 result.push_back({flow, cost});
251 prev_cost_per_flow = d;
252 }
253 return result;
254 }
255
256 3.2 max flow
257
258 namespace internal {
259 template <class T>
260 struct simple_queue {
261     std::vector<T> payload;
262     int pos = 0;
263     void reserve(int n) { payload.reserve(n);
264     }
265     int size() const { return int(payload.size
266         ()) - pos; }
267     bool empty() const { return pos == int(
268         payload.size()); }
269     void push(const T& t) { payload.push_back(
270         t); }
271     T& front() { return payload[pos]; }
272     void clear() {
273         payload.clear();
274         pos = 0;
275     }
276     void pop() { pos++; }
277 };
278 // namespace internal
279
280 template <class Cap>
281 struct MaxFlowGraph {
282 public:
283     MaxFlowGraph() : MaxFlowGraph(0) {}
284     explicit MaxFlowGraph(int n) : _n(n), g(n)
285         {}
286
287     int add_edge(int from, int to, Cap cap) {
288         assert(0 <= from && from < _n);
289         assert(0 <= to && to < _n);
290         assert(0 <= cap);
291         int m = int(pos.size());
292         pos.push_back({from, int(g[from].size()
293             )});
294         int from_id = int(g[from].size());

```

```

34 int to_id = int(g[to].size());
35 if (from == to) to_id++;
36 g[from].push_back(_edge{to, to_id, cap})
37 ;
38 g[to].push_back(_edge{from, from_id, 0})
39 ;
40 return m;
41 }
42 struct edge {
43     int from, to;
44     Cap cap, flow;
45 };
46 edge get_edge(int i) {
47     int m = int(pos.size());
48     assert(0 <= i && i < m);
49     auto _e = g[pos[i].first][pos[i].second
50 ];
51     auto _re = g[_e.to][_e.rev];
52     return edge{pos[i].first, _e.to, _e.cap
53 + _re.cap, _re.cap};
54 }
55 std::vector<edge> edges() {
56     int m = int(pos.size());
57     std::vector<edge> result;
58     for (int i = 0; i < m; i++) {
59         result.push_back(get_edge(i));
60     }
61     return result;
62 }
63 void change_edge(int i, Cap new_cap, Cap
64 new_flow) {
65     int m = int(pos.size());
66     assert(0 <= i && i < m);
67     assert(0 <= new_flow && new_flow <=
68 new_cap);
69     auto& _e = g[pos[i].first][pos[i].second
70 ];
71     auto& _re = g[_e.to][_e.rev];
72     _e.cap = new_cap - new_flow;
73     _re.cap = new_flow;
74 }
75 Cap flow(int s, int t) { return flow(s, t,
76 std::numeric_limits<Cap>::max()); }
77 Cap flow(int s, int t, Cap flow_limit) {
78     assert(0 <= s && s < _n);
79     assert(0 <= t && t < _n);
80     assert(s != t);
81     std::vector<int> level(_n), iter(_n);
82     internal::simple_queue<int> que;
83     auto bfs = [&]() {
84         std::fill(level.begin(), level.end(),
85 -1);
86         level[s] = 0;
87         que.clear();
88         que.push(s);
89         while (!que.empty()) {
90             int v = que.front();
91             que.pop();
92             for (auto e : g[v]) {
93                 if (e.cap == 0 || level[e.to] >=
94 level[v] + 1) continue;
95                 level[e.to] = level[v] + 1;
96                 if (e.to == t) return;
97                 que.push(e.to);
98             }
99         }
100     };
101     auto dfs = [&](auto self, int v, Cap up)
102 {
103     if (v == s) return up;
104     Cap res = 0;
105     int level_v = level[v];
106     for (int& i = iter[v]; i < int(g[v].
107 size()); i++) {
108         _edge& e = g[v][i];
109         if (level_v <= level[e.to] || g[e.to
110 ][e.rev].cap == 0) continue;
111         Cap d = self(self, e.to, std::min(up
112 - res, g[e.to][e.rev].cap));
113         if (d <= 0) continue;
114         g[v][i].cap -= d;
115         g[e.to][e.rev].cap += d;
116         res += d;
117         if (res == up) return res;
118     }
119     level[v] = _n;
120     return res;
121 };
122 Cap flow = 0;
123 while (flow < flow_limit) {
124     bfs();
125     if (level[t] == -1) break;
126     std::fill(iter.begin(), iter.end(), 0)
127 ;
128     Cap f = dfs(dfs, t, flow_limit - flow)
129 ;
130     if (!f) break;
131     flow += f;
132 }
133 return flow;
134 }
135 std::vector<bool> min_cut(int s) {
136     std::vector<bool> visited(_n);
137     internal::simple_queue<int> que;
138     que.push(s);
139     while (!que.empty()) {
140         int p = que.front();
141         que.pop();
142         visited[p] = true;
143         for (auto e : g[p]) {
144             if (e.cap && !visited[e.to]) {
145                 visited[e.to] = true;
146                 que.push(e.to);
147             }
148         }
149     }
150     return visited;
151 }
152 private:
153     int _n;
154     struct _edge {
155         int to, rev;
156         Cap cap;
157     };

```

4 Graph

4.1 tarjan

```

1 int dfn[N], low[N], dfncnt, s[N], in_stack[N]
2 , tp;
3 int scc[N], sc; // 结点 i 所在 SCC 的编号
4 int sz[N]; // 强连通 i 的大小
5 void tarjan(int u) {
6     low[u] = dfn[u] = ++dfncnt, s[++tp] = u,
7     in_stack[u] = 1;
8     for (int i = h[u]; i; i = e[i].nex) {
9         const int& v = e[i].t;
10        if (!dfn[v]) {
11            tarjan(v);
12            low[u] = min(low[u], low[v]);
13        } else if (in_stack[v]) {
14            low[u] = min(low[u], dfn[v]);
15        }
16    }
17    if (dfn[u] == low[u]) {
18        ++sc;
19        while (s[tp] != u) {
20            scc[s[tp]] = sc;
21            sz[sc]++;
22            in_stack[s[tp]] = 0;
23            --tp;
24        }
25        scc[s[tp]] = sc;
26        sz[sc]++;
27        in_stack[s[tp]] = 0;
28        --tp;
29    }

```

4.2 heavy light decomposition

```

1 const int kMax = 1e5 + 5;
2 int n, m, r, kMod, op, x, y, z, cnt = 0;
3 int a[kMax] = {}, dep[kMax] = {}, fa[kMax] =
4 {}, siz[kMax] = {},
5 hson[kMax] = {},
6 int nid[kMax] = {}, na[kMax] = {}, ltop[kMax]
7 = {},
8 seg[kMax << 2] = {}, tag[kMax << 2] =
9 {};
10 vector<int> g[kMax];
11 void dfs1(int u, int f) {
12     dep[u] = dep[f] + 1;
13     fa[u] = f;
14     siz[u] = 1;

```

```

13 for (auto v : g[u]) {
14     if (v == f) continue;
15     dfs1(v, u);
16     siz[u] += siz[v];
17     if (siz[v] > siz[hson[u]]) hson[u] = v;
18 }
19 }
20 void dfs2(int u, int cur_ltop) {
21     nid[u] = ++cnt;
22     na[cnt] = a[u];
23     ltop[u] = cur_ltop;
24     if (hson[u]) dfs2(hson[u], cur_ltop);
25     for (auto v : g[u]) {
26         if (v == fa[u] || v == hson[u]) continue
27 ;
28         dfs2(v, v);
29     }
30 }
31 void build(int u, int l, int r) {
32     if (l == r) {
33         seg[u] = na[l] % kMod;
34         return;
35     }
36     int mid = (l + r) >> 1;
37     build(u << 1, l, mid);
38     build(u << 1 | 1, mid + 1, r);
39     seg[u] = (seg[u << 1] + seg[u << 1 | 1]) %
40 kMod;
41 }
42 void push_down(int u, int l, int r) {
43     if (tag[u]) {
44         int mid = (l + r) >> 1;
45         seg[u << 1] = (seg[u << 1] + tag[u] * (
46 mid - l + 1)) % kMod;
47         seg[u << 1 | 1] = (seg[u << 1 | 1] + tag
48 [u] * (r - mid)) % kMod;
49         tag[u << 1] = (tag[u << 1] + tag[u]) %
50 kMod;
51         tag[u << 1 | 1] = (tag[u << 1 | 1] + tag
52 [u]) % kMod;
53         tag[u] = 0;
54     }
55 }
56 int query(int u, int l, int r, int ql, int
57 qr) {
58     if (ql <= l && r <= qr) return seg[u];
59     push_down(u, l, r);
60     int mid = (l + r) >> 1, res = 0;
61     if (ql <= mid) res = (res + query(u << 1,
62 l, mid, ql, qr)) % kMod;
63     if (qr > mid) res = (res + query(u << 1 |
64 1, mid + 1, r, ql, qr)) % kMod;
65     return res;
66 }
67 void update(int u, int l, int r, int ql, int
68 qr, int k) {
69     if (ql <= l && r <= qr) {
70         seg[u] = (seg[u] + k * (r - l + 1)) %
71 kMod;
72         tag[u] = (tag[u] + k) % kMod;
73         return;

```

```

68 }
69 push_down(u, 1, r);
70 int mid = (l + r) >> 1;
71 if (ql <= mid) update(u << 1, 1, mid, ql,
72   qr, k);
73 if (qr > mid) update(u << 1 | 1, mid + 1,
74   r, ql, qr, k);
75 seg[u] = (seg[u << 1] + seg[u << 1 | 1]) %
76   kMod;
77 }
78 int query_path(int u, int v) {
79   int res = 0;
80   while (ltop[u] != ltop[v]) {
81     if (dep[ltop[u]] < dep[ltop[v]]) swap(u,
82       v);
83     res = (res + query(1, 1, n, nid[ltop[u]
84       ], nid[u])) % kMod;
85     u = fa[ltop[u]];
86   }
87   if (dep[u] > dep[v]) swap(u, v);
88   res = (res + query(1, 1, n, nid[u], nid[v
89     ])) % kMod;
90   return res;
91 }
92 void update_path(int u, int v, int k) {
93   k %= kMod;
94   while (ltop[u] != ltop[v]) {
95     if (dep[ltop[u]] < dep[ltop[v]]) swap(u,
96       v);
97     update(1, 1, n, nid[ltop[u]], nid[u], k)
98     ;
99     u = fa[ltop[u]];
100   }
101   if (dep[u] > dep[v]) swap(u, v);
102   update(1, 1, n, nid[u], nid[v], k);
103 }
104 int query_subtree(int u) { return query(1,
105   1, n, nid[u], nid[u] + siz[u] - 1); }
106 void update_subtree(int u, int k) {
107   update(1, 1, n, nid[u], nid[u] + siz[u] -
108     1, k);
109 }
110 int32_t main() {
111   cin.tie(nullptr)->sync_with_stdio(false);
112   cin >> n >> m >> r >> kMod;
113   for (int i = 1; i <= n; i++) cin >> a[i];
114   for (int i = 1; i < n; i++) {
115     cin >> x >> y;
116     g[x].push_back(y);
117     g[y].push_back(x);
118   }
119   dfs1(r, 0);
120   dfs2(r, r);
121   build(1, 1, n);
122   while (m--) {
123     cin >> op >> x;
124     if (op == 1) {
125       cin >> y >> z;
126       update_path(x, y, z);

```

```

127   } else if (op == 2) {
128     cin >> y;
129     cout << query_path(x, y) << '\n';
130   } else if (op == 3) {
131     cin >> z;
132     update_subtree(x, z);
133   } else {
134     cout << query_subtree(x) << '\n';
135   }
136 }
137 return 0;

```

4.3 tree centroid

```

1 // 这份代码默认节点编号从 1 开始，即 i ∈ [1,
2   n]
3 int size[MAXN], // 这个节点的「大小」（所有
4   子树上节点数 + 该节点）
5 weight[MAXN], // 这个节点的「重量」，即
6   所有子树「大小」的最大值
7 centroid[2]; // 用于记录树的重心（存的
8   是节点编号）
9 void GetCentroid(int cur, int fa) { // cur
10   表示当前节点（current）
11   size[cur] = 1;
12   weight[cur] = 0;
13   for (int i = head[cur]; i != -1; i = e[i].
14     nxt) {
15     if (e[i].to != fa) { // e[i].to 表示这
16       条有向边所通向的节点。
17       GetCentroid(e[i].to, cur);
18       size[cur] += size[e[i].to];
19       weight[cur] = max(weight[cur], size[e
20     [i].to]);
21   }
22   weight[cur] = max(weight[cur], n - size[
23     cur]);
24   if (weight[cur] <= n / 2) { // 依照树的重
25     心的定义统计
26     centroid[centroid[0] != 0] = cur;
27   }
28 }

```

5 Number Theory

5.1 bit set

```

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 }

```

5.2 basic

```

1 template<typename T>
2 void gcd(const T &a,const T &b,T &d,T &x,T &
3   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 long long int phi[N+1];
8 void phiTable(){
9   for(int i=1;i<=N;i++)phi[i]=i;
10  for(int i=1;i<=N;i++)for(x=i*2;x<=N;x+=i)
11    phi[x]-=phi[i];
12 }
13 void all_divdown(const LL &n) { // all n/x
14   for(LL a=1;a<=n;a=n/(n/(a+1))) {
15     // dosomething;
16   }
17 }
18 const int MAXPRIME = 1000000;
19 int iscom[MAXPRIME], prime[MAXPRIME],
20   primecnt;
21 int phi[MAXPRIME], mu[MAXPRIME];
22 void sieve(void){
23   memset(iscom,0,sizeof(iscom));
24   primecnt = 0;
25   phi[1] = mu[1] = 1;
26   for(int i=2;i<MAXPRIME;i++) {
27     if(!iscom[i]) {
28       prime[primecnt++] = i;
29       mu[i] = -1;
30       phi[i] = i-1;
31     }
32     for(int j=0;j<primecnt;j++) {
33       int k = i * prime[j];
34       if(k>MAXPRIME) break;
35       iscom[k] = prime[j];
36       if(i%prime[j]==0) {
37         mu[k] = 0;
38         phi[k] = phi[i] * prime[j];
39         break;
40       } else {
41         mu[k] = -mu[i];
42         phi[k] = phi[i] * (prime[j]-1);
43       }
44     }
45   }
46 }
47 bool g_test(const LL &g, const LL &p, const
48   vector<LL> &v) {
49   for(int i=0;i<v.size();i++)
50     if(modexp(g,(p-1)/v[i],p)==1)

```

```

51   return false;
52   return true;
53 }
54 LL primitive_root(const LL &p) {
55   if(p==2) return 1;
56   vector<LL> v;
57   Factor(p-1,v);
58   v.erase(unique(v.begin(), v.end()), v.end
59     ());
60   for(LL g=2;g<p;g++)
61     if(g_test(g,p,v))
62       return g;
63   puts("primitive_root NOT FOUND");
64   return -1;
65 }
66 int Legendre(const LL &a, const LL &p) {
67   return modexp(a%p,(p-1)/2,p); }
68 LL inv(const LL &a, const LL &n) {
69   LL d,x,y;
70   gcd(a,n,d,x,y);
71   return d==1 ? (x+n)%n : -1;
72 }
73 int inv[maxn];
74 LL invtable(int n,LL P){
75   inv[1]=1;
76   for(int i=2;i<n;i++)
77     inv[i]=(P-(P/i))*inv[P%i]%P;
78 }
79 LL log_mod(const LL &a, const LL &b, const
80   LL &p) {
81   // a ^ x = b ( mod p )
82   int m=sqrt(p+.5), e=1;
83   LL v=inv(modexp(a,m,p), p);
84   map<LL,int> x;
85   x[1]=0;
86   for(int i=1;i<m;i++) {
87     e = LLmul(e,a,p);
88     if(!x.count(e)) x[e] = i;
89   }
90   for(int i=0;i<m;i++) {
91     if(x.count(b)) return i*m + x[b];
92     b = LLmul(b,v,p);
93   }
94   return -1;
95 }
96 LL Tonelli-Shanks(const LL &n, const LL &p)
97 {
98   // x^2 = n ( mod p )
99   if(n==0) return 0;
100  if(Legendre(n,p)!=1) while(1) { puts("SQRT
101    ROOT does not exist"); }
102  int S = 0;
103  LL Q = p-1;
104  while( !(Q&1) ) { Q>>=1; ++S; }
105  if(S==1) return modexp(n%p,(p+1)/4,p);
106  LL z = 2;
107  for(; Legendre(z,p)!=-1; z++)
108    LL c = modexp(z,Q,p);
109  LL R = modexp(n%p,(Q+1)/2,p), t = modexp(n
110    %p,Q,p);
111  int M = S;
112  while(1) {

```



```

108 if(t==1) return R;
109 LL b = modexp(c, 1L<<(M-i-1), p);
110 R = LLMul(R, b, p);
111 t = LLMul(LLmul(b, b, p), t, p);
112 c = LLMul(b, b, p);
113 M = i;
114 }
115 return -1;
116 }
117
118 template<typename T>
119 T Euler(T n){
120     T ans=n;
121     for(T i=2; i<=n; ++i){
122         if(n%i==0){
123             ans=ans/i*(i-1);
124             while(n%i==0)n/=i;
125         }
126     }
127     if(n>1)ans=ans/n*(n-1);
128     return ans;
129 }
130
131 //Chinese_remainder_theorem
132 template<typename T>
133 T pow_mod(T n, T k, T m){
134     T ans=1;
135     for(n=(n>=m?n%m:n); k>=1){
136         if(k&1)ans=ans*n%m;
137         n=n*n%m;
138     }
139     return ans;
140 }
141
142 template<typename T>
143 T crt(vector<T> &m, vector<T> &a){
144     T M=1, tM, ans=0;
145     for(int i=0; i<(int)m.size(); ++i)M*=m[i];
146     for(int i=0; i<(int)a.size(); ++i){
147         tM=M/m[i];
148         ans=(ans+(a[i]*tM%M)*pow_mod(tM, Euler(m[i])-1, m[i])%M)%M;
149         /*如果m[i]是質數·Euler(m[i])-1=m[i]-2·
150         就不用算Euler了*/
151     }
152     return ans;
153 }
154
155 //java code
156 //求sqrt(N)的連分數
157 public static void Pell(int n){
158     BigInteger N, p1, p2, q1, q2, a0, a1, a2, g1, g2, h1, h2, p, q;
159     g1=q2=p1=BigInteger.ZERO;
160     h1=q1=p2=BigInteger.ONE;
161     a0=a1=BigInteger.valueOf((int)Math.sqrt(1.0*n));
162     BigInteger ans=a0.multiply(a0);
163     if(ans.equals(BigInteger.valueOf(n))){
164         System.out.println("No solution!");
165         return;
166     }
167     while(true){
168         g2=a1.multiply(h1).subtract(g1);
169         h2=N.subtract(g2.pow(2)).divide(h1);
170         a2=g2.add(a0).divide(h2);

```

```

169 p=a1.multiply(p2).add(p1);
170 q=a1.multiply(q2).add(q1);
171 if(p.pow(2).subtract(N.multiply(q.pow(2))).compareTo(BigInteger.ONE)==0)
172     break;
173 g1=g2; h1=h2; a1=a2;
174 p1=p2; p2=p;
175 q1=q2; q2=q;
176 }
177 System.out.println(p+" "+q);

```

5.3 數位統計

```

1 ll d[65], dp[65][2]; //up區間是不是完整
2 ll dfs(int p, bool is8, bool up){
3     if(!p) return 1; // 回傳0是不是答案
4     if(!up&&~dp[p][is8]) return dp[p][is8];
5     int mx = up?d[p]:9; //可以用的有那些
6     ll ans=0;
7     for(int i=0; i<=mx; ++i){
8         if(is8&&i==7) continue;
9         ans += dfs(p-1, i==8, up&&i==mx);
10    }
11    if(!up) dp[p][is8]=ans;
12    return ans;
13 }
14 ll f(ll N){
15     int k=0;
16     while(N){ // 把數字先分解到陣列
17         d[++k] = N%10;
18         N/=10;
19     }
20     return dfs(k, false, true);
21 }

```

5.4 find real root

```

1 // an*x^n + ... + a1x + a0 = 0;
2 int sign(double x){
3     return x < -eps ? -1 : x > eps;
4 }
5
6 double get(const vector<double>&coef, double x){
7     double e = 1, s = 0;
8     for(auto i : coef) s += i*e, e *= x;
9     return s;
10 }
11
12 double find(const vector<double>&coef, int n, double lo, double hi){
13     double sign_lo, sign_hi;
14     if(!sign_lo = sign(get(coef, lo))) return lo;
15     if(!sign_hi = sign(get(coef, hi))) return hi;
16     if(sign_lo * sign_hi > 0) return INF;

```

```

17 for(int stp = 0; stp < 100 && hi - lo > eps; ++stp){
18     double m = (lo+hi)/2.0;
19     int sign_mid = sign(get(coef, m));
20     if(!sign_mid) return m;
21     if(sign_lo*sign_mid < 0) hi = m;
22     else lo = m;
23 }
24 return (lo+hi)/2.0;
25 }
26
27 vector<double> cal(vector<double>coef, int n){
28     vector<double>res;
29     if(n == 1){
30         if(sign(coef[1])) res.pb(-coef[0]/coef[1]);
31         return res;
32     }
33     vector<double>dcoef(n);
34     for(int i = 0; i < n; ++i) dcoef[i] = coef[i+1]*(i+1);
35     vector<double>droot = cal(dcoef, n-1);
36     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], droot[i+1]);
40         if(tmp < INF) res.pb(tmp);
41     }
42     return res;
43 }
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 }

```

5.5 中國剩餘定理

```

1 int exgcd(int a, int b, int &x, int &y){
2     int x1 = 1, x2 = 0, x3 = 0, x4 = 1;
3     while(b != 0){
4         int c = a / b;
5         std::tie(x1, x2, x3, x4, a, b) =
6             std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
7     }
8     x = x1, y = x2;
9     return a;
10 }
11
12 LL CRT(int k, LL* a, LL* r){
13     LL n = 1, ans = 0;
14     for(int i = 1; i <= k; ++i) n = n * r[i];
15     for(int i = 1; i <= k; ++i){
16         LL m = n / r[i], b, y;
17         exgcd(m, r[i], b, y); // b * m mod r[i] = 1
18         ans = (ans + a[i] * m * b % n) % n;
19     }
20     return (ans % n + n) % n;

```

5.6 質因數分解

```

1 LL func(const LL n, const LL mod, const int c){
2     return (LLmul(n, n, mod) + c + mod) % mod;
3 }
4
5 LL pollrho(const LL n, const int c) { // 循環節長度
6     LL a=1, b=1;
7     a=func(a, n, c)%n;
8     b=func(b, n, c)%n; b=func(b, n, c)%n;
9     while(gcd(abs(a-b), n) != 1) {
10         a=func(a, n, c)%n;
11         b=func(b, n, c)%n; b=func(b, n, c)%n;
12     }
13     return gcd(abs(a-b), n);
14 }
15
16 void prefactor(LL &n, vector<LL> &v){
17     for(int i=0; i<12; ++i){
18         while(n%prime[i]==0){
19             v.push_back(prime[i]);
20             n/=prime[i];
21         }
22     }
23 }
24
25 void smallfactor(LL n, vector<LL> &v){
26     if(n<MAXPRIME){
27         while(isp[(int)n]){
28             v.push_back(isp[(int)n]);
29             n/=isp[(int)n];
30         }
31         v.push_back(n);
32     } else {
33         for(int i=0; i<primecnt&&prime[i]*prime[i]<=n; ++i){
34             while(n%prime[i]==0){
35                 v.push_back(prime[i]);
36                 n/=prime[i];
37             }
38         }
39         if(n!=1) v.push_back(n);
40     }
41 }
42
43 void comfactor(const LL &n, vector<LL> &v){
44     if(n<1e9){
45         smallfactor(n, v);
46         return;
47     }
48     if(Isprime(n)){
49         v.push_back(n);
50         return;
51     }
52     LL d;
53     for(int c=3; ++c){
54         d = pollrho(n, c);
55         if(d!=n) break;
56     }
57     comfactor(d, v);
58     comfactor(n/d, v);
59 }
60 }

```

```

61 void Factor(const LL &x, vector<LL> &v) {
62     LL n = x;
63     if(n==1) { puts("Factor 1"); return; }
64     prefactor(n,v);
65     if(n==1) return;
66     comfactor(n,v);
67     sort(v.begin(),v.end());
68 }
69
70 void AllFactor(const LL &n,vector<LL> &v) {
71     vector<LL> tmp;
72     Factor(n,tmp);
73     v.clear();
74     v.push_back(1);
75     int len;
76     LL now=1;
77     for(int i=0;i<tmp.size();++i) {
78         if(i==0 || tmp[i]!=tmp[i-1]) {
79             len = v.size();
80             now = 1;
81         }
82         now*=tmp[i];
83         for(int j=0;j<len;++j)
84             v.push_back(v[j]*now);
85     }
86 }

```

5.7 Matrix

```

1 template<typename T>
2 struct Matrix{
3     using rt = std::vector<T>;
4     using mt = std::vector<rt>;
5     using matrix = Matrix<T>;
6     int r,c;
7     mt m;
8     Matrix(int r,int c):r(r),c(c),m(r,rt(c)){}
9     rt& operator[](int i){return m[i];}
10    matrix operator+(const matrix &a){
11        matrix rev(r,c);
12        for(int i=0;i<r;++i)
13            for(int j=0;j<c;++j)
14                rev[i][j]=m[i][j]+a.m[i][j];
15        return rev;
16    }
17    matrix operator-(const matrix &a){
18        matrix rev(r,c);
19        for(int i=0;i<r;++i)
20            for(int j=0;j<c;++j)
21                rev[i][j]=m[i][j]-a.m[i][j];
22        return rev;
23    }
24    matrix operator*(const matrix &a){
25        matrix rev(r,a.c);
26        matrix tmp(a.c,a.r);
27        for(int i=0;i<a.r;++i)
28            for(int j=0;j<a.c;++j)
29                tmp[j][i]=a.m[i][j];
30        for(int i=0;i<r;++i)
31            for(int j=0;j<a.c;++j)
32                for(int k=0;k<c;++k)
33                    rev.m[i][j]+=m[i][k]*tmp[j][k];
34        return rev;

```

```

35    }
36    bool inverse(){
37        Matrix t(r,r+c);
38        for(int y=0;y<r;y++){
39            t.m[y][c+y] = 1;
40            for(int x=0;x<c;++x)
41                t.m[y][x]=m[y][x];
42        }
43        if( !t.gas() )
44            return false;
45        for(int y=0;y<r;y++){
46            for(int x=0;x<c;++x)
47                m[y][x]=t.m[y][c+x]/t.m[y][y];
48        }
49        return true;
50    }
51    T gas(){
52        vector<T> lazy(r,1);
53        bool sign=false;
54        for(int i=0;i<r;++i){
55            if( m[i][i]==0 ){
56                int j=i+1;
57                while(j<r&&!m[j][i])j++;
58                if(j==r)continue;
59                m[i].swap(m[j]);
60                sign=!sign;
61            }
62            for(int j=0;j<r;++j){
63                if(i==j)continue;
64                lazy[j]=lazy[j]*m[i][i];
65                T mx=m[j][i];
66                for(int k=0;k<c;++k)
67                    m[j][k]=m[j][k]*m[i][i]-m[i][k]*mx;
68            }
69        }
70        T det=sign?-1:1;
71        for(int i=0;i<r;++i){
72            det = det*m[i][i];
73            det = det/lazy[i];
74            for(auto &j:m[i])j/=lazy[i];
75        }
76        return det;
77    };

```

5.8 FFT

```

1 template<typename T,typename VT=vector<
2     complex<T> > >
3 struct FFT{
4     const T pi;
5     FFT(const T pi=acos((T)-1)):pi(pi){}
6     unsigned bit_reverse(unsigned a,int len){
7         a=((a&0x55555555U)<<1)|((a&0xAAAAAAAAU)>>1);
8         a=((a&0x33333333U)<<2)|((a&0xCCCCCCCCU)>>2);
9         a=((a&0x0F0F0F0FU)<<4)|((a&0xFF0F0F0FU)>>4);
10        a=((a&0x00FF00FFU)<<8)|((a&0xFFFF00FFU)>>8);
11        a=((a&0x0000FFFFU)<<16)|((a&0xFFFF0000U)>>16);
12        return a>>(32-len);
13    }
14    void fft(bool is_inv,VT &in,VT &out,int N)
15    {

```

```

14        int bitlen=__lg(N),num=is_inv?-1:1;
15        for(int i=0;i<N;++i)out[bit_reverse(i,
16            bitlen)]=in[i];
17        for(int step=2;step<=N;step<=1){
18            const int mh=step>>1;
19            for(int i=0;i<mh;++i){
20                complex<T> wi=exp(complex<T>(0,i*num
21                    *pi/mh));
22                for(int j=i;j<N;j+=step){
23                    int k=j+mh;
24                    complex<T> u=out[j],t=wi*out[k];
25                    out[j]=u+t;
26                    out[k]=u-t;
27                }
28            }
29        }
30        if(is_inv)for(int i=0;i<N;++i)out[i]/=N;

```

5.9 linear sieve

```

1 const int N = 10000000;
2 vector<int> lp(N+1);
3 vector<int> pr;
4
5 for (int i=2; i <= N; ++i) {
6     if (lp[i] == 0) {
7         lp[i] = i;
8         pr.push_back(i);
9     }
10    for (int j = 0; i * pr[j] <= N; ++j) {
11        lp[i * pr[j]] = pr[j];
12        if (pr[j] == lp[i]) {
13            break;
14        }
15    }
16 }

```

6 String

6.1 KMP

```

1 /*產生fail function*/
2 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
12     fail)*/
13 int kmp_match(char *A,int lenA,char *B,int
14     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, id=fail[id];
19         }
20     }
21     return ans;
22 }

```

6.2 suffix array lcp

```

1 #define radix_sort(x,y){\
2     for(i=0;i<A;++i)c[i]=0;\
3     for(i=0;i<n;++i)c[x[y[i]]]++;\
4     for(i=1;i<A;++i)c[i]+=c[i-1];\
5     for(i=n-1;~i;--i)sa[--c[x[y[i]]]]=y[i];\
6 }
7 #define AC(r,a,b){\
8     r[a]!r[b]||a+k>n||r[a+k]!r[b+k]\
9 void suffix_array(const char *s,int n,int *
10     sa,int *rank,int *tmp,int *c){
11     int A='z'+1,i,k,id=0;
12     for(i=0;i<n;++i)rank[tmp[i]=i]=s[i];
13     radix_sort(rank,tmp);
14     for(k=1;id<n-1;k<=1){
15         for(id=0,i=n-k;i<n;++i)tmp[id++]=i;
16         for(i=0;i<n;++i)
17             if(sa[i]>k)tmp[id++]=(sa[i]-k);
18         radix_sort(rank,tmp);
19         swap(rank,tmp);
20         for(rank[sa[0]]=id=0,i=1;i<n;++i)
21             rank[sa[i]]=id+=AC(tmp,sa[i-1],sa[i]);
22         A=id+1;
23     }
24 }
25 //高度數組 sa:後綴數組 rank:排名
26 void suffix_array_lcp(const char *s,int len,
27     int *h,int *sa,int *rank){
28     for(int i=0;i<len;++i)rank[sa[i]]=i;
29     for(int i=0,k=0;i<len;++i){
30         if(rank[i]==0)continue;
31         if(k)--k;
32         while(s[i+k]==s[sa[rank[i]-1]+k])++k;
33         h[rank[i]]=k;
34     }
35     h[0]=0; // h[k]=Lcp(sa[k],sa[k-1]);

```

6.3 Z

```

1 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
8             [i];

```

```

7   if(i+z[i]-1>r)r=i+z[i]-1,l=i;
8   }
9   }

```

6.4 hash

```

1  #define MAXN 1000000
2  #define mod 1073676287
3  /*mod 必須要是質數*/
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)%mod*/
8  void hash_init(int len,T prime){
9      h_base[0]=1;
10     for(int i=1;i<=len;++i){
11         h[i]=(h[i-1]*prime+s[i-1])%mod;
12         h_base[i]=(h_base[i-1]*prime)%mod;
13     }
14 }
15 T get_hash(int l,int r){/*閉區間寫法，設編號
    為 0 ~ len-1*/
16     return (h[r+1]-(h[l]*h_base[r-l+1])%mod+
17         mod)%mod;

```

6.5 AC 自動機

```

1  template<char L='a',char R='z'>
2  class ac_automaton{
3      struct joe{
4          int next[R-L+1],fail,efl,ed,cnt_dp,vis;
5          joe():ed(0),cnt_dp(0),vis(0){
6              for(int i=0;i<=R-L;++i)next[i]=0;
7          }
8      };
9      public:
10         std::vector<joe> S;
11         std::vector<int> q;
12         int qs,qe,vt;
13         ac_automaton():S(1),qs(0),qe(0),vt(0){}
14         void clear(){
15             q.clear();
16             S.resize(1);
17             for(int i=0;i<=R-L;++i)S[0].next[i]=0;
18             S[0].cnt_dp=S[0].vis=qs=qe=vt=0;
19         }
20         void insert(const char *s){
21             int o=0;
22             for(int i=0,id;s[i];++i){
23                 id=s[i]-L;
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             }
30             ++S[o].ed;
31         }

```

```

32 void build_fail(){
33     S[0].fail=S[0].efl=-1;
34     q.clear();
35     q.push_back(0);
36     ++qe;
37     while(qs!=qe){
38         int pa=q[qs++],id,t;
39         for(int i=0;i<=R-L;++i){
40             t=S[pa].next[i];
41             if(!t)continue;
42             id=S[pa].fail;
43             while(~id&&!S[id].next[i])id=S[id].fail;
44             S[t].fail=~id?S[id].next[i]:0;
45             S[t].efl=S[S[t].fail].ed?S[t].fail:S[t].fail].efl;
46             q.push_back(t);
47             ++qe;
48         }
49     }
50 }
51 /*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)*/
52 int match_0(const char *s){
53     int ans=0,id,p=0,i;
54     for(i=0;s[i];++i){
55         id=s[i]-L;
56         while(!S[p].next[id]&&p=S[p].fail;
57             if(!S[p].next[id])continue;
58         p=S[p].next[id];
59         ++S[p].cnt_dp;/*匹配成功則它所有後綴都
            可以被匹配(DP 計算)*/
60     }
61     for(i=qe-1;i>=0;--i){
62         ans+=S[q[i]].cnt_dp*S[q[i]].ed;
63         if(~S[q[i]].fail)S[S[q[i]].fail].cnt_dp+=S[q[i]].cnt_dp;
64     }
65     return ans;
66 }
67 /*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)*/
68 int match_1(const char *s)const{
69     int ans=0,id,p=0,t;
70     for(int i=0;s[i];++i){
71         id=s[i]-L;
72         while(!S[p].next[id]&&p=S[p].fail;
73             if(!S[p].next[id])continue;
74         p=S[p].next[id];
75         if(S[p].ed)ans+=S[p].ed;
76         for(t=S[p].efl;~t;t=S[t].efl){
77             ans+=S[t].ed;/*因為都走efl邊所以保證
                匹配成功*/
78         }
79     }
80     return ans;
81 }
82 /*枚舉(s的子字串a)的所有相異字串各恰一次
    並傳回次數 O(N*M^(1/3))*/
83 int match_2(const char *s){
84     int ans=0,id,p=0,t;
85     ++vt;
86     /*把戳記 vt+=1，只要 vt 沒溢位，所有 S[p].
        vis==vt 就會變成 false

```

```

87     這種利用 vt 的方法可以 O(1) 歸零 vis 陣列*/
88     for(int i=0;s[i];++i){
89         id=s[i]-L;
90         while(!S[p].next[id]&&p=S[p].fail;
91             if(!S[p].next[id])continue;
92         p=S[p].next[id];
93         if(S[p].ed&&S[p].vis!=vt){
94             S[p].vis=vt;
95             ans+=S[p].ed;
96         }
97         for(t=S[p].efl;~t&&S[t].vis!=vt;t=S[t].efl){
98             S[t].vis=vt;
99             ans+=S[t].ed;/*因為都走efl邊所以保證
                匹配成功*/
100        }
101    }
102    return ans;
103 }
104 /*把 AC 自動機變成真的自動機*/
105 void evolution(){
106     for(qs=1;qs!=qe;){
107         int p=q[qs++];
108         for(int i=0;i<=R-L;++i)
109             if(S[p].next[i]==0)S[p].next[i]=S[S[p].fail].next[i];
110     }
111 }
112 }

```

6.6 minimal string rotation

```

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);/*最小循環表示法起始位置
14 }

```

6.7 manacher

```

1 //原字串: asdsasdsa
2 //先把字串變成這樣: @#a#s#d#s#a#s#d#s#a#
3 void manacher(char *s,int len,int *z){
4     int l=0,r=0;
5     for(int i=1;i<len;++i){
6         z[i]=r>i?min(z[2*i-1],r-i):1;
7         while(s[i+z[i]]==s[i-z[i]])++z[i];
8         if(z[i]+i>r)r=z[i]+i,l=i;
9     }/*ans = max(z)-1
10 }

```

6.8 reverseBWT

```

1 const int MAXN = 305, MAXC = 'Z';
2 int ranks[MAXN], tots[MAXC], first[MAXC];
3 void rankBWT(const string &bw){
4     memset(ranks,0,sizeof(int)*bw.size());
5     memset(tots,0,sizeof(tots));
6     for(size_t i=0;i<bw.size();++i)
7         ranks[i] = tots[int(bw[i])]+1;
8 }
9 void firstCol(){
10     memset(first,0,sizeof(first));
11     int totc = 0;
12     for(int c='A';c<='Z';++c){
13         if(!tots[c]) continue;
14         first[c] = totc;
15         totc += tots[c];
16     }
17 }
18 string reverseBwt(string bw,int begin){
19     rankBWT(bw), firstCol();
20     int i = begin; /*原字串最後一個元素的位置
21     string res;
22     do{
23         char c = bw[i];
24         res = c + res;
25         i = first[int(c)] + ranks[i];
26     }while( i != begin );
27     return res;
28 }

```

7 Tarjan

7.1 橋連通分量

```

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 void add_edge(int u,int v){
13     G[u].push_back(E.size());
14     E.emplace_back(u,v);
15     G[v].push_back(E.size());
16     E.emplace_back(v,u);
17 }
18 void dfs(int u,int re=-1){/*u 當前點，re 為 u 連
    接前一個點的邊
19     int v;
20     low[u]=vis[u]=++Time;
21     st[top++]=u;
22     for(int e:G[u]){

```

```

23 v=E[e].v;
24 if(!vis[v]){
25     dfs(v,e^1);//e^1 反向邊
26     low[u]=min(low[u],low[v]);
27     if(vis[u]<low[v]){
28         E[e].is_bridge=E[e^1].is_bridge=1;
29         ++bridge_cnt;
30     }
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 void bcc_init(int n){
41     Time=bcc_cnt=bridge_cnt=top=0;
42     E.clear();
43     for(int i=1;i<=n;++i){
44         G[i].clear();
45         vis[i]=bcc_id[i]=0;
46     }
47 }

```

7.2 dominator tree

```

1 struct dominator_tree{
2     static const int MAXN=5005;
3     int n;// 1-base
4     vector<int> G[MAXN], rG[MAXN];
5     int pa[MAXN], dfn[MAXN], id[MAXN], dfnCnt;
6     int semi[MAXN], idom[MAXN], best[MAXN];
7     vector<int> tree[MAXN]; // tree here
8     void init(int _n){
9         n = _n;
10        for(int i=1; i<=n; ++i)
11            G[i].clear(), rG[i].clear();
12    }
13    void add_edge(int u, int v){
14        G[u].push_back(v);
15        rG[v].push_back(u);
16    }
17    void dfs(int u){
18        id[dfn[u]=++dfnCnt]=u;
19        for(auto v:G[u]) if(!dfn[v])
20            dfs(v), pa[dfn[v]]=dfn[u];
21    }
22    int find(int y, int x){
23        if(y <= x) return y;
24        int tmp = find(pa[y], x);
25        if(semi[best[y]] > semi[best[pa[y]]])
26            best[y] = best[pa[y]];
27        return pa[y] = tmp;
28    }
29    void tarjan(int root){
30        dfnCnt = 0;
31        for(int i=1; i<=n; ++i){
32            dfn[i] = idom[i] = 0;
33            tree[i].clear();

```

```

34        best[i] = semi[i] = i;
35    }
36    dfs(root);
37    for(int i=dfnCnt; i>1; --i){
38        int u = id[i];
39        for(auto v:rG[u]) if(v=dfn[v]){
40            find(v, i);
41            semi[i]=min(semi[i], semi[best[v]]);
42        }
43        tree[semi[i]].push_back(i);
44        for(auto v:tree[pa[i]]){
45            find(v, pa[i]);
46            idom[v] = semi[best[v]]==pa[i]
47                ? pa[i] : best[v];
48        }
49        tree[pa[i]].clear();
50    }
51    for(int i=2; i<=dfnCnt; ++i){
52        if(idom[i] != semi[i])
53            idom[i] = idom[idom[i]];
54        tree[id[idom[i]]].push_back(id[i]);
55    }
56 }
57 }dom;

```

7.3 雙連通分量 & 割點

```

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
7 int st[N], top;
8 void dfs(int u, int pa=-1){//u當前點 · pa父親
9     int t, child=0;
10    low[u]=vis[u]=++Time;
11    st[top++]=u;
12    for(int v:G[u]){
13        if(!vis[v]){
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                do{
20                    bcc_id[t=st[--top]]=bcc_cnt;
21                    bcc[bcc_cnt].push_back(t);
22                }while(t!=v);
23                bcc_id[u]=bcc_cnt;
24                bcc[bcc_cnt].push_back(u);
25            }
26        }else if(vis[v]<vis[u]&&v!=pa){//反向邊
27            low[u] = min(low[u], vis[v]);
28        }//u是dfs樹的根要特判
29        if(pa!=-1&&child<2)is_cut[u]=0;
30    }
31    void bcc_init(int n){
32        Time=bcc_cnt=top=0;
33        for(int i=1; i<=n; ++i){
34            G[i].clear();
35            is_cut[i]=vis[i]=bcc_id[i]=0;

```

```

36 }
37 }

```

7.4 tnfshb017 2 sat

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define MAXN 8001
4 #define MAXN2 MAXN*4
5 #define n(X) ((X)+2*N)
6 vector<int> v[MAXN2], rv[MAXN2], vis_t;
7 int N, M;
8 void addedge(int s, int e){
9     v[s].push_back(e);
10    rv[e].push_back(s);
11 }
12 int scc[MAXN2];
13 bool vis[MAXN2]={false};
14 void dfs(vector<int> *uv, int n, int k=-1){
15     vis[n]=true;
16     for(int i=0; i<uv[n].size(); ++i)
17         if(!vis[uv[n][i]])
18             dfs(uv, uv[n][i], k);
19     if(uv==v)vis_t.push_back(n);
20     scc[n]=k;
21 }
22 void solve(){
23     for(int i=1; i<=N; ++i){
24         if(!vis[i])dfs(v, i);
25         if(!vis[n(i)])dfs(v, n(i));
26     }
27     memset(vis, 0, sizeof(vis));
28     int c=0;
29     for(int i=vis_t.size()-1; i>=0; --i)
30         if(!vis[vis_t[i]])
31             dfs(rv, vis_t[i], c++);
32 }
33 int main(){
34     int a, b;
35     scanf("%d%d", &N, &M);
36     for(int i=1; i<=N; ++i){
37         // (A or B)&(!A & !B) A^B
38         a=i*2-1;
39         b=i*2;
40         addedge(n(a), b);
41         addedge(n(b), a);
42         addedge(a, n(b));
43         addedge(b, n(a));
44     }
45     while(M--){
46         scanf("%d%d", &a, &b);
47         a = a>0?a*2-1:-a*2;
48         b = b>0?b*2-1:-b*2;
49         // A or B
50         addedge(n(a), b);
51         addedge(n(b), a);
52     }
53     solve();
54     bool check=true;
55     for(int i=1; i<=2*N; ++i)
56         if(scc[i]==scc[n(i)])
57             check=false;
58     if(check){

```

```

59     printf("%d\n", N);
60     for(int i=1; i<=2*N; i+=2){
61         if(scc[i]>scc[i+2*N]) putchar('+');
62         else putchar('-');
63     }
64     puts("");
65 }else puts("0");
66 return 0;
67 }

```

8 other

8.1 WhatDay

```

1 int whatday(int y, int m, int d){
2     if(m<=2)m+=12, --y;
3     if(y<1752||y==1752&&m<9||y==1752&&m==9&&d<3)
4         return (d+2*m+3*(m+1)/5+y+y/4+5)%7;
5     return (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7;
6 }

```

8.2 最大矩形

```

1 LL max_rectangle(vector<int> s){
2     stack<pair<int, int> > st;
3     st.push(make_pair(-1, 0));
4     s.push_back(0);
5     LL ans=0;
6     for(size_t i=0; i<s.size(); ++i){
7         int h=s[i];
8         pair<int, int> now=make_pair(h, i);
9         while(h<st.top().first){
10            now=st.top();
11            st.pop();
12            ans=max(ans, (LL)(i-now.second)*now.first);
13        }
14        if(h>st.top().first){
15            st.push(make_pair(h, now.second));
16        }
17    }
18    return ans;
19 }

```

8.3 上下最大正方形

```

1 void solve(int n, int a[], int b[]){// 1-base
2     int ans=0;
3     deque<int> da, db;
4     for(int l=1, r=1; r<=n; ++r){
5         while(da.size()&&a[da.back()]>a[r]){
6             da.pop_back();
7         }

```



```

8 | da.push_back(r);
9 | while(db.size() && b[db.back()] >= b[r]){
10 |     db.pop_back();
11 | }
12 | db.push_back(r);
13 | for(int d=a[da.front()+b[db.front()]]; r-
14 |     1+1>d; ++1){
15 |     if(da.front()==1) da.pop_front();
16 |     if(db.front()==1) db.pop_front();
17 |     if(da.size() && db.size()){
18 |         d=a[da.front()+b[db.front()]];
19 |     }
20 |     ans=max(ans, r-1+1);
21 | }
22 | printf("%d\n", ans);
23 | }

```

(h) $\text{ans} = \min_cut(S, T)$ (i) $|E| = 0$ 要特殊判斷

7. 弦圖：

(a) 點數大於 3 的環都要有一條弦

(b) 完美消除序列從後往前依次給每個點染色，給每個點染上可以染的最小顏色

(c) 最大團大小 = 色數

(d) 最大獨立集：完美消除序列從前往後能選就選

(e) 最小團覆蓋：最大獨立集的點和他延伸的邊構成

(f) 區間圖是弦圖

(g) 區間圖的完美消除序列：將區間按造又端點由小到大的排序

(h) 區間圖染色：用線段樹做

9.1.3 dinic 特殊圖複雜度

1. 單位流： $O\left(\min\left(V^{3/2}, E^{1/2}\right)E\right)$ 2. 二分圖： $O\left(V^{1/2}E\right)$

9.1.4 0-1 分數規劃

 $x_i \in \{0, 1\}$ ， x_i 可能會有其他限制，求 $\max\left(\frac{\sum B_i x_i}{\sum C_i x_i}\right)$ 1. $D(i, g) = B_i - g \times C_i$ 2. $f(g) = \sum D(i, g)x_i$ 3. $f(g) = 0$ 時 g 為最佳解， $f(g) < 0$ 沒有意義4. 因為 $f(g)$ 單調可以二分搜 g

5. 或用 Dinkelbach 通常比較快

```

1 | binary_search(){
2 |     while(r-l>eps){
3 |         g=(l+r)/2;
4 |         for(i:所有元素) D[i]=B[i]-g*C[i]; //D(i,g)
5 |         找出一組合法 x[i] 使 f(g) 最大;
6 |         if(f(g)>0) l=g;
7 |         else r=g;
8 |     }
9 |     Ans = r;
10 | }
11 | Dinkelbach(){
12 |     g=任意狀態(通常設為0);
13 |     do{
14 |         Ans=g;
15 |         for(i:所有元素) D[i]=B[i]-g*C[i]; //D(i,g)
16 |         找出一組合法 x[i] 使 f(g) 最大;
17 |         p=0, q=0;
18 |         for(i:所有元素)
19 |             if(x[i]) p+=B[i], q+=C[i];
20 |         g=p/q; //更新解，注意 q=0 的情況
21 |     } while(abs(Ans-g)>EPS);
22 |     return Ans;
23 | }

```

9 zformula

9.1 formula

9.1.1 Pick 公式

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

9.1.2 圖論

- 對於平面圖， $F = E - V + C + 1$ ， C 是連通分量數
- 對於平面圖， $E \leq 3V - 6$
- 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 $Cv(G)$ ，最小邊覆蓋設為 $Ce(G)$ 。對於任意連通圖：

- $I(G) + Cv(G) = |V|$
- $M(G) + Ce(G) = |V|$

- 對於連通二分圖：

- $I(G) = Cv(G)$
- $M(G) = Ce(G)$

- 最大權閉合圖：

- $C(u, v) = \infty, (u, v) \in E$
- $C(S, v) = W_v, W_v > 0$
- $C(v, T) = -W_v, W_v < 0$
- $\text{ans} = \sum_{W_v > 0} W_v - \text{flow}(S, T)$

- 最大密度子圖：

- 求 $\max\left(\frac{w_e + w_v}{|V'|}\right), e \in E', v \in V'$

(b) $U = \sum_{v \in V} 2W_v + \sum_{e \in E} W_e$ (c) $C(u, v) = W_{(u, v)}, (u, v) \in E$ ，雙向邊(d) $C(S, v) = U, v \in V$ (e) $D_u = \sum_{(u, v) \in E} W_{(u, v)}$ (f) $C(v, T) = U + 2g - D_v - 2W_v, v \in V$ (g) 二分搜 g ： $l = 0, r = U, \text{eps} = 1/n^2$ if $(U \times |V| - \text{flow}(S, T)) / 2 > 0$ $l = \text{mid}$ else $r = \text{mid}$

9.1.5 學長公式

- $\sum_{d|n} \phi(n) = n$
- $g(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) \times g(n/d)$
- Harmonic series $H_n = \ln(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$
- $\gamma = 0.57721566490153286060651209008240243104215$
- 格雷碼 = $n \oplus (n >> 1)$
- $SG(A + B) = SG(A) \oplus SG(B)$
- 選轉矩陣 $M(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

9.1.6 基本數論

- $\sum_{d|n} \mu(n) = [n == 1]$
- $g(m) = \sum_{d|m} f(d) \Leftrightarrow f(m) = \sum_{d|m} \mu(d) \times g(m/d)$
- $\sum_{i=1}^n \sum_{j=1}^m \text{互質數量} = \sum \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$
- $\sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j) = n \sum_{d|n} d \times \phi(d)$

9.1.7 排組公式

- k 卡特蘭 $\frac{C_n^{kn}}{n(k-1)+1} \cdot C_m^n = \frac{n!}{m!(n-m)!}$
- $H(n, m) \cong x_1 + x_2 \dots + x_n = k, \text{num} = C_k^{n+k-1}$
- Stirling number of $2^n d, n$ 人分 k 組方法數目
 - $S(0, 0) = S(n, n) = 1$
 - $S(n, 0) = 0$
 - $S(n, k) = kS(n-1, k) + S(n-1, k-1)$
- Bell number, n 人分任意多組方法數目
 - $B_0 = 1$
 - $B_n = \sum_{i=0}^n S(n, i)$
 - $B_{n+1} = \sum_{k=0}^n C_k^n B_k$
 - $B_{p+n} \equiv B_n + B_{n+1} \pmod{p}$, p is prime
 - $B_{p^m+n} \equiv mB_n + B_{n+1} \pmod{p}$, p is prime
 - From $B_0 : 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975$

- Derangement, 錯排，沒有人在自己位置上

- $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \dots + (-1)^n \frac{1}{n!})$
- $D_n = (n-1)(D_{n-1} + D_{n-2}), D_0 = 1, D_1 = 0$
- From $D_0 : 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496$

- Binomial Equality

- $\sum_k \binom{r}{m+k} \binom{s}{n-k} = \binom{r+s}{m+n}$
- $\sum_k \binom{m}{m+k} \binom{s}{n-k} = \binom{l+s}{l+n}$
- $\sum_k \binom{m}{m+k} \binom{s}{n-k} (-1)^k = (-1)^{l+m} \binom{s-m}{n-l}$
- $\sum_{k \leq l} \binom{l-k}{m} \binom{s}{k-n} (-1)^k = (-1)^{l+m} \binom{s-m-1}{l-n-m}$
- $\sum_{0 \leq k \leq l} \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1}$
- $\binom{r}{k} = (-1)^k \binom{k-r-1}{k}$

- $\binom{r}{m} \binom{m}{k} = \binom{r}{k} \binom{r-k}{m-k}$
- $\sum_{k \leq n} \binom{r+k}{k} = \binom{r+n+1}{n}$
- $\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n+1}{m+1}$
- $\sum_{k \leq m} \binom{m+r}{k} x^k y^k = \sum_{k \leq m} \binom{-r}{k} (-x)^k (x+y)^{m-k}$

9.1.8 冪次，冪次和

- $a^{b\%p} P = a^{b\% \varphi(p) + \varphi(p)}, b \geq \varphi(p)$
- $1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
- $1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
- $1^5 + 2^5 + 3^5 + \dots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
- $0^k + 1^k + 2^k + \dots + n^k = P(k), P(k) = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P(0) = n+1$
- $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^n C_k^{n+1} B_k m^{n+1-k}$
- $\sum_{j=0}^m C_j^{m+1} B_j = 0, B_0 = 1$
- 除了 $B_1 = -1/2$ ，剩下的奇數項都是 0
- $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330$

9.1.9 Burnside's lemma

- $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
- $X^g = t^{c(g)}$
- G 表示有幾種轉法， X^g 表示在那種轉法下，有幾種是會保持對稱的， t 是顏色數， $c(g)$ 是循環節不動的面數。
- 正立方體塗三顏色，轉 0 有 3^6 個元素不變，轉 90 有 6 種，每種有 3^3 不變，180 有 $3 \times 3^4 \cdot 120$ (角) 有 $8 \times 3^2 \cdot 180$ (邊) 有 6×3^3 ，全部 $\frac{1}{24} (3^6 + 6 \times 3^3 + 3 \times 3^4 + 8 \times 3^2 + 6 \times 3^3) = \frac{57}{57}$

9.1.10 Count on a tree

- Rooted tree: $s_{n+1} = \frac{1}{n} \sum_{i=1}^n (i \times a_i \times \sum_{j=1}^{\lfloor n/i \rfloor} a_{n+1-i \times j})$
- Unrooted tree:
 - Odd: $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$
 - Even: $Odd + \frac{1}{2} a_{n/2} (a_{n/2} + 1)$
- Spanning Tree

- 完全圖 $n^n - 2$
- 一般圖 (Kirchhoff's theorem) $M[i][i] = \text{degree}(V_i), M[i][j] = -1, \text{if have } E(i, j), 0 \text{ if no edge. Delete any one row and col in } A, \text{ans} = \det(A)$

Contents

7.3	雙連通分量 & 割點	12
7.4	tnfshb017 2 sat	12
8	other	12
8.1	WhatDay	12
8.2	最大矩形	12
8.3	上下最大正方形	12
9	zformula	13
9.1	formula	13
9.1.1	Pick 公式	13
9.1.2	圖論	13
9.1.3	dinic 特殊圖複雜度	13
9.1.4	0-1 分數規劃	13
9.1.5	學長公式	13
9.1.6	基本數論	13
9.1.7	排組公式	13
9.1.8	冪次, 冪次和	13
9.1.9	Burnside's lemma	13
9.1.10	Count on a tree	13

ACM ICPC Judge Test - Angry Crow Takes Flight!

C++ Resource Test

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 namespace system_test {
5
6     const size_t KB = 1024;
7     const size_t MB = KB * 1024;
8     const size_t GB = MB * 1024;
```

```
9     size_t block_size, bound;
10     void stack_size_dfs(size_t depth = 1) {
11         if (depth >= bound)
12             return;
13         int8_t ptr[block_size]; // 若無法編譯將
14             block_size 改成常數
15         memset(ptr, 'a', block_size);
16         cout << depth << endl;
17         stack_size_dfs(depth + 1);
18     }
19
20     void stack_size_and_runtime_error(size_t
21         block_size, size_t bound = 1024) {
22         system_test::block_size = block_size;
23         system_test::bound = bound;
24         stack_size_dfs();
25     }
26
27     double speed(int iter_num) {
28         const int block_size = 1024;
29         volatile int A[block_size];
30         auto begin = chrono::high_resolution_clock
31             ::now();
32         while (iter_num--)
33             for (int j = 0; j < block_size; ++j)
34                 A[j] += j;
35         auto end = chrono::high_resolution_clock::
36             now();
```

```
37         chrono::duration<double> diff = end -
38             begin;
39         return diff.count();
40     }
41
42     void runtime_error_1() {
43         // Segmentation fault
44         int *ptr = nullptr;
45         *(ptr + 7122) = 7122;
46     }
47
48     void runtime_error_2() {
49         // Segmentation fault
50         int *ptr = (int *)memset;
51         *ptr = 7122;
52     }
53
54     void runtime_error_3() {
55         // munmap_chunk(): invalid pointer
56         int *ptr = (int *)memset;
57         delete ptr;
58     }
59
60     void runtime_error_4() {
61         // free(): invalid pointer
62         int *ptr = new int[7122];
63         ptr += 1;
64         delete[] ptr;
```

```
65
66         chrono::duration<double> diff = end -
67             begin;
68         return diff.count();
69     }
70
71     void runtime_error_5() {
72         // maybe illegal instruction
73         int a = 7122, b = 0;
74         cout << (a / b) << endl;
75     }
76
77     void runtime_error_6() {
78         // floating point exception
79         volatile int a = 7122, b = 0;
80         cout << (a / b) << endl;
81     }
82
83     void runtime_error_7() {
84         // call to abort.
85         assert(false);
86     }
87
88     // namespace system_test
89
90     #include <sys/resource.h>
91     void print_stack_limit() { // only work in
92         Linux
93         struct rlimit l;
94         getrlimit(RLIMIT_STACK, &l);
95         cout << "stack_size = " << l.rlim_cur << "
96             byte" << endl;
```