

1 Computational Geometry

1.1 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(x),p2(y){}
39     void pton() const{ //轉成一般式
40         a=p1.y-p2.y;
41         b=p2.x-p1.x;
42         c=-a*p1.x-b*p1.y;
43     }
44     T ori(const point<T> &p) const{ //點和有向直
45         線的關係 · >0左邊 · =0在線上 <0右邊
46         return (p2-p1).cross(p-p1);
47     }
48     T btw(const point<T> &p) const{ //點投影落在
49         線段上 <=0
50         return (p1-p).dot(p2-p);
51     }
52     bool point_on_segment(const point<T>&p)
53         const{ //點是否在線段上
54         return ori(p)==0&&btw(p)<=0;
55     }
56     T dis2(const point<T> &p, bool is_segment
57         =0) const{ //點跟直線/線段的距離平方
58         point<T> v=p2-p1,v1=v-p1;
59         if(is_segment){
60             point<T> v2=p-p1;
61             if(v.dot(v1)<=0) return v1.abs2();
62             if(v.dot(v2)>=0) return v2.abs2();
63         }
64         T tmp=v.cross(v1);
65         return tmp*tmp/v.abs2();
66     }
67     T seg_dis2(const line<T> &l) const{ //兩線段
68         距離平方
69         return min({dis2(l.p1,1),dis2(l.p2,1),l.
70             dis2(p1,1),l.dis2(p2,1)});
71     }
72     point<T> projection(const point<T> &p)
73         const{ //點對直線的投影
74         point<T> n=(p2-p1).normal();
75         return p-n*(p-p1).dot(n)/n.abs2();
76     }
77     point<T> mirror(const point<T> &p) const{
78         //點對直線的鏡射 · 要先呼叫pton轉成一般式
79         point<T> R;
80         T d=a*b+b*b;
81         R.x=(b*b*p.x-a*a*p.x-2*a*b*p.y-2*a*c)/d;
82         R.y=(a*a*p.y-b*b*p.y-2*a*b*p.x-2*b*c)/d;
83         return R;
84     }
85     bool equal(const line &l) const{ //直線相等
86         return ori(l.p1)==0&&ori(l.p2)==0;
87     }
88     bool parallel(const line &l) const{
89         return (p1-p2).cross(l.p1-l.p2)==0;
90     }
91     bool cross_seg(const line &l) const{
92         return (p2-p1).cross(l.p1-p1)*(p2-p1).
93             cross(l.p2-p1)<=0; //直線是否交線段
94     }
95     int line_intersect(const line &l) const{ //
96         直線相交情況 · -1無限多點 · 1交於一點 · 0
97         不相交
98         return parallel(l)?(ori(l.p1)==0?-1:0)
99             :1;
100     }
101     int seg_intersect(const line &l) const{
102         T c1=ori(l.p1), c2=ori(l.p2);
103         T c3=l.ori(p1), c4=l.ori(p2);
104         if(c1==0&&c2==0){ //共線
105             bool b1=btw(l.p1)>=0,b2=btw(l.p2)>=0;
106             T a3=1.btw(p1),a4=1.btw(p2);
107             if(b1&&b2&&a3==0&&a4==0) return 2;
108             if(b1&&b2&&a3>=0&&a4==0) return 3;
109             if(b1&&b2&&a3>=0&&a4>=0) return 0;
110             return -1; //無限交點
111         }else if(c1*c2<=0&&c3*c4<=0) return 1;
112         return 0; //不相交
113     }
114     point<T> line_intersection(const line &l)
115         const{ //直線交點 */
116         point<T> a=p2-p1,b=l.p2-l.p1,s=l.p1-p1;
117         //if(a.cross(b)==0) return INF;
118         return p1+a*(s.cross(b)/a.cross(b));
119     }
120     point<T> seg_intersection(const line &l)
121         const{ //線段交點
122         int res=seg_intersect(l);
123         if(res<=0) assert(0);
124         if(res==2) return p1;
125         if(res==3) return p2;
126         return line_intersection(l);
127     }
128 };
129 template<typename T>
130 struct polygon{
131     polygon(){}
132     vector<point<T> > p; //逆時針順序
133     T area() const{ //面積
134         T ans=0;
135         for(int i=p.size()-1,j=0;j<(int)p.size()
136             ;i=j++){
137             ans+=p[i].cross(p[j]);
138         }
139         return ans/2;
140     }
141     point<T> center_of_mass() const{ //重心
142         T cx=0,cy=0,w=0;
143         for(int i=p.size()-1,j=0;j<(int)p.size()
144             ;i=j++){
145             T a=p[i].cross(p[j]);
146             cx+=(p[i].x+p[j].x)*a;
147             cy+=(p[i].y+p[j].y)*a;
148             w+=a;
149         }
150         return point<T>(cx/3/w,cy/3/w);
151     }
152     char ahas(const point<T>& t) const{ //點是否
153         在簡單多邊形內 · 是的話回傳1 · 在邊上回
154         傳-1 · 否則回傳0
155         bool c=0;
156         for(int i=0,j=p.size()-1;i<p.size();j=i
157             ++){
158             if(line<T>(p[i],p[j]).point_on_segment
159                 (t)) return -1;
160             else if((p[i].y>t.y)!=p[j].y>t.y)&&
161                 t.x<(p[j].x-p[i].x)*(t.y-p[i].y)/(p[j
162                     ].y-p[i].y)+p[i].x)
163                 c=!c;
164             return c;
165         }
166     }
167     char point_in_convex(const point<T>&x)
168         const{
169         int l=1,r=(int)p.size()-2;
170         while(l<r){ //點是否在凸多邊形內 · 是的話
171             回傳1 · 在邊上回傳-1 · 否則回傳0
172             int mid=(l+r)/2;
173             T a1=(p[mid]-p[0]).cross(x-p[0]);
174             T a2=(p[mid+1]-p[0]).cross(x-p[0]);
175             if(a1>=0&&a2<=0){
176                 T res=(p[mid+1]-p[mid]).cross(x-p[
177                     mid]);
178                 return res>0?-1:(res==0?-1:0);
179             }else if(a1<0) r=mid-1;
180             else l=mid+1;
181         }
182         return 0;
183     }
184     vector<T> getA() const{ //凸包邊對x軸的夾角
185     vector<T> res; //一定是遞增的
186     for(size_t i=0;i<p.size();i++){
187         res.push_back((p[(i+1)%p.size()]-p[i])
188             .getA());
189     }
190     return res;
191 }
192 bool line_intersect(const vector<T>&A,
193     const line<T> &l) const{ //0(LogN)
194     int f1=upper_bound(A.begin(),A.end(),(l.
195         p1-l.p2).getA())-A.begin();
196     int f2=upper_bound(A.begin(),A.end(),(l.
197         p2-l.p1).getA())-A.begin();
198     return l.cross_seg(line<T>(p[f1],p[f2]));
199 }
200 polygon cut(const line<T> &l) const{ //凸包
201     對直線切割 · 得到直線L左側的凸包
202     polygon ans;
203     for(int n=p.size(),i=n-1,j=0;j<n;i=j++){
204         if(l.ori(p[i])>=0){
205             ans.p.push_back(p[i]);
206             if(l.ori(p[j])<0)
207                 ans.p.push_back(l.
208                     line_intersection(line<T>(p[i
209                         ],p[j])));
210             }else if(l.ori(p[j])>0)
211                 ans.p.push_back(l.line_intersection(
212                     line<T>(p[i],p[j])));
213         }
214     }
215     return ans;
216 }
217 static bool monotone_chain_cmp(const point
218     <T>& a, const point<T>& b){ //凸包排序函
219     數
220     return (a.x<b.x)|| (a.x==b.x&&a.y<b.y);
221 }
222 void monotone_chain(vector<point<T> > &s){
223     //凸包
224     sort(s.begin(),s.end(),
225         monotone_chain_cmp);
226     p.resize(s.size()+1);
227     int m=0;
228     for(size_t i=0;i<s.size();i++){
229         while(m>=2&&(p[m-1]-p[m-2]).cross(s[i
230             ]-p[m-2])<=0)--m;
231         p[m++]=s[i];
232     }
233     for(int i=s.size()-2,t=m+1;i>=0;--i){
234         while(m>=t&&(p[m-1]-p[m-2]).cross(s[i
235             ]-p[m-2])<=0)--m;
236         p[m++]=s[i];
237     }
238     if(s.size()>1)--m;
239     p.resize(m);
240 }
241 T diam() const{ //直徑
242     int n=p.size(),t=1;
243     T ans=0;p.push_back(p[0]);
244     for(int i=0;i<n;i++){
245         point<T> now=p[i+1]-p[i];
246         while(now.cross(p[t+1]-p[i])>now.cross
247             (p[t]-p[i])) t=(t+1)%n;
248         ans=max(ans,(p[i]-p[t]).abs2());
249     }
250     return p.pop_back(),ans;
251 }
252 T min_cover_rectangle() const{ //最小覆蓋矩形

```

```

211 int n=p.size(),t=1,r=1,l;
212 if(n<3)return 0;//也可以做最小周長矩形
213 T ans=1e99;p.push_back(p[0]);
214 for(int i=0;i<n;i++){
215     point<T> now=p[i+1]-p[i];
216     while(now.cross(p[t+1]-p[i])>now.cross
217           (p[t]-p[i]))t=(t+1)%n;
218     while(now.dot(p[r+1]-p[i])>now.dot(p[r
219           ]-p[i]))r=(r+1)%n;
220     if(!i)l=r;
221     while(now.dot(p[l+1]-p[i])<=now.dot(p[
222           l]-p[i]))l=(l+1)%n;
223     T d=now.abs2();
224     T tmp=now.cross(p[t]-p[i])*(now.dot(p[
225           r]-p[i])-now.dot(p[l]-p[i]))/d;
226     ans=min(ans,tmp);
227 }
228 return p.pop_back(),ans;
229
230 T dis2(polygon &p1){//凸包最近距離平方
231 vector<point<T> > &P=p,&Q=p1.p;
232 int n=P.size(),m=Q.size(),l=0,r=0;
233 for(int i=0;i<n;++i)if(P[i].y<P[l].y)l=i;
234 for(int i=0;i<m;++i)if(Q[i].y<Q[r].y)r=i;
235 P.push_back(P[0]),Q.push_back(Q[0]);
236 T ans=1e99;
237 for(int i=0;i<n;++i){
238     while((P[l]-P[l+1]).cross(Q[r+1]-Q[r])
239           <0)r=(r+1)%m;
240     ans=min(ans,line<T>(P[l],P[l+1]).
241           seg_dis2(line<T>(Q[r],Q[r+1])));
242     l=(l+1)%n;
243 }
244 return P.pop_back(),Q.pop_back(),ans;
245
246 static char sign(const point<T>&t){
247     return (t.y==0?t.x:t.y)<0;
248 }
249 static bool angle_cmp(const line<T>& A,
250     const line<T>& B){
251     point<T> a=A.p2-A.p1,b=B.p2-B.p1;
252     return sign(a)<sign(b)||((sign(a)==sign(b)
253     )&&a.cross(b)>0);
254 }
255 int halfplane_intersection(vector<line<T>
256     > &s){//半平面交
257     sort(s.begin(),s.end(),angle_cmp);//線段
258     左側為該線段半平面
259     int L,R,n=s.size();
260     vector<point<T> > px(n);
261     vector<line<T> > q(n);
262     q[L=R=0]=s[0];
263     for(int i=1;i<n;++i){
264         while(L<R&&s[i].ori(px[R-1])<=0)--R;
265         while(L<R&&s[i].ori(px[L])<=0)++L;
266         q[++R]=s[i];
267         if(q[R].parallel(q[R-1])){
268             --R;
269             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
270         }
271         if(L<R)px[R-1]=q[R-1].
272             line_intersection(q[R]);
273     }
274     while(L<R&&q[L].ori(px[R-1])<=0)--R;
275     p.clear();
276
277 if(R-L<=1)return 0;
278 px[R]=q[R].line_intersection(q[L]);
279 for(int i=L;i<R;++i)p.push_back(px[i]);
280 return R-L+1;
281 }
282
283 template<typename T>
284 struct triangle{
285     point<T> a,b,c;
286     triangle(){
287         triangle(const point<T> &a,const point<T>
288             &b,const point<T> &c):a(a),b(b),c(c){
289             T area()const{
290                 T t=(b-a).cross(c-a)/2;
291                 return t>0?t:-t;
292             }
293     point<T> barycenter()const{//重心
294         return (a+b+c)/3;
295     }
296     point<T> circumcenter()const{//外心
297         static line<T> u,v;
298         u.p1=(a+b)/2;
299         u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-
300             b.x);
301         v.p1=(a+c)/2;
302         v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-
303             c.x);
304         return u.line_intersection(v);
305     }
306     point<T> incenter()const{//內心
307         T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2
308             ()),C=sqrt((a-b).abs2());
309         return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+
310             B*b.y+C*c.y)/(A+B+C);
311     }
312     point<T> perpercenter()const{//垂心
313         return barycenter()*3-circumcenter()*2;
314     }
315 }
316
317 template<typename T>
318 struct point3D{
319     T x,y,z;
320     point3D(){
321         point3D(const T&x,const T&y,const T&z):x(x
322             ),y(y),z(z){
323             point3D operator+(const point3D &b)const{
324                 return point3D(x+b.x,y+b.y,z+b.z);
325             }
326             point3D operator-(const point3D &b)const{
327                 return point3D(x-b.x,y-b.y,z-b.z);
328             }
329             point3D operator*(const T &b)const{
330                 return point3D(x*b,y*b,z*b);
331             }
332             point3D operator/(const T &b)const{
333                 return point3D(x/b,y/b,z/b);
334             }
335             bool operator==(const point3D &b)const{
336                 return x==b.x&&y==b.y&&z==b.z;
337             }
338             T dot(const point3D &b)const{
339                 return x*b.x+y*b.y+z*b.z;
340             }
341             point3D cross(const point3D &b)const{
342                 return point3D(y*b.z-z*b.y,z*b.x-x*b.z,
343                     x*b.y-y*b.x);
344             }
345             T abs2()const{//向量長度的平方
346                 return dot(*this);
347             }
348             T area2(const point3D &b)const{//和b、原點
349                 圍成面積的平方
350                 return cross(b).abs2()/4;
351             }
352 };
353
354 template<typename T>
355 struct line3D{
356     point3D<T> p1,p2;
357     line3D(){
358         line3D(const point3D<T> &p1,const point3D<
359             T> &p2):p1(p1),p2(p2){
360             T dis2(const point3D<T> &p,bool is_segment
361                 =0)const{//點跟直線/線段的距離平方
362                 point3D<T> v=p2-p1,v1=p-p1;
363                 if(is_segment){
364                     point3D<T> v2=p-p2;
365                     if(v.dot(v1)<=0)return v1.abs2();
366                     if(v.dot(v2)>=0)return v2.abs2();
367                 }
368                 point3D<T> tmp=v.cross(v1);
369                 return tmp.abs2()/v.abs2();
370             }
371     pair<point3D<T>,point3D<T> > closest_pair(
372         const line3D<T> &l1,const
373         point3D<T> v1=(p1-p2),v2=(l.p1-l.p2);
374         point3D<T> N=v1.cross(v2),ab(p1-l.p1);
375         //if(N.abs2()==0)return NULL;平行或重合
376         T tmp=N.dot(ab),ans=tmp*tmp/N.abs2();//
377         最近點對距離
378         point3D<T> d1=p2-p1,d2=l.p2-l.p1,D=d1.
379             cross(d2),G=l.p1-p1;
380         T t1=(G.cross(d2)).dot(D)/D.abs2();
381         T t2=(G.cross(d1)).dot(D)/D.abs2();
382         return make_pair(p1+d1*t1,l.p1+d2*t2);
383     }
384     bool same_side(const point3D<T> &a,const
385         point3D<T> &b)const{
386         return (p2-p1).cross(a-p1).dot((p2-p1).
387             cross(b-p1))>0;
388     }
389 };
390
391 template<typename T>
392 struct plane{
393     point3D<T> p0,n;//平面上的點和法向量
394     plane(){
395         plane(const point3D<T> &p0,const point3D<T>
396             &n):p0(p0),n(n){
397             T dis2(const point3D<T> &p)const{//點到平
398                 面距離的平方
399                 T tmp=(p-p0).dot(n);
400                 return tmp*tmp/n.abs2();
401             }
402             point3D<T> projection(const point3D<T> &p)
403                 const{
404                 return p-n*(p-p0).dot(n)/n.abs2();
405             }
406             point3D<T> line_intersection(const line3D<
407                 T> &l1)const{
408                 T tmp=n.dot(l.p2-l.p1);//等於0表示平行或
409                 重合該平面
410                 return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/
411                     tmp);
412             }
413             line3D<T> plane_intersection(const plane &
414                 p1)const{
415                 point3D<T> e=n.cross(p1.n),v=n.cross(e);
416                 T tmp=p1.n.dot(v);//等於0表示平行或重合
417                 該平面
418                 point3D<T> q=p0+(v*(p1.n.dot(p1.p0-p0))/
419                     tmp);
420                 return line3D<T>(q,q+e);
421             }
422         }
423     };
424
425 1.2 MinCircleCover
426
427 1 const double eps = 1e-10;
428 2 int sign(double a){
429 3     return fabs(a)<eps?0:a>0?1:-1;
430 4 }
431 5 template<typename T>
432 6 T len(point<T> p){
433 7     return sqrt(p.dot(p));
434 8 }
435 9 template<typename T>
436 10 point<T> findCircumcenter(point<T> A,point<T>
437     B,point<T> C){
438 11     point<T> AB = B-A;
439 12     point<T> AC = C-A;
440 13     T AB_len_sq = AB.x*AB.x+AB.y*AB.y;
441 14     T AC_len_sq = AC.x*AC.x+AC.y*AC.y;
442 15     T D = AB.x*AC.y-AB.y*AC.x;
443 16     T X = A.x+(AC.y*AB_len_sq-AB.y*AC_len_sq)
444         /(2*D);
445 17     T Y = A.y+(AB.x*AC_len_sq-AC.x*AB_len_sq)
446         /(2*D);
447     return point<T>(X,Y);
448 18 }
449 19
450 20 template<typename T>
451 21 pair<T, point<T>> MinCircleCover(vector<
452     point<T>> &p){
453 22 // 回傳最小覆蓋圓{半徑, 中心}
454 23 random_shuffle(p.begin(),p.end());
455 24 int n = p.size();
456 25 point<T> c = p[0]; T r = 0;
457 26 for(int i=1;i<n;i++){
458 27     if(sign(len(c-p[i])-r) > 0){ // 不在圓內
459 28         c = p[i], r = 0;
460 29         for(int j=0;j<i;j++){
461 30             if(sign(len(c-p[j])-r) > 0) {
462 31                 c = (p[i]+p[j])/2.0;
463 32                 r = len(c-p[i]);
464 33                 for(int k=0;k<j;k++){
465 34                     if(sign(len(c-p[k])-r) > 0){
466 35                         //c=triangle<T>(p[i],p[j],p[k]).
467                             circumcenter();
468                         c = findCircumcenter(p[i],p[j]
469                             ,p[k]);
470                         r = len(c-p[i]);
471                     }
472                 }
473             }
474         }
475     }
476     return make_pair(r,c);
477 }

```

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(
7         mid,R));
8     inplace_merge(L, mid, R, ycmp);
9     static vector<point> b; b.clear();
10    for(auto u=L;u<R;++u){
11        if((u->x-x)*(u->x-x)>=d) continue;
12        for(auto v=b.rbegin();v!=b.rend();++v){
13            T dx=u->x-v->x, dy=u->y-v->y;
14            if(dy*dy>=d) break;
15            d=min(d,dx*dx+dy*dy);
16        }
17        b.push_back(*u);
18    }
19    return d;
20 }
21 T closest_pair(vector<point<T>> &v){
22     sort(v.begin(),v.end(),xcmp);
23     return closest_pair(v.begin(),v.end());
24 }

```

2 DP

2.1 basic DP

```

1 // 0/1背包問題
2 for(int i=0;i<n;i++){
3     for(int k=W;k>=w[i];k--){
4         dp[k] = max(dp[k],dp[k-w[i]]+v[i]);
5     }
6     //因為不能重複拿，所以要倒回來
7 }
8 //無限背包問題
9 dp[0] = 1;
10 for(int i=0;i<n;i++){
11     int a;cin>>a;
12     for(int k=a;k<=m;k++){
13         dp[k] += dp[k-a];
14         if(dp[k]>=mod) dp[k] -= mod;
15     }
16 }
17 //LIS問題
18 for(int i=0;i<n;i++){
19     cin>>x;
20     auto it = lower_bound(dp.begin(),dp.end()
21         (),x);
22     if(it == dp.end()){
23         dp.emplace_back(x);
24     }
25     else {
26         *it = x;
27     }
28 }

```

```

28 cout<<dp.size();
29 //LCS問題
30 signed main() {
31     string a,b;
32     cin>>a>>b;
33     vector<vector<int>> dp(a.size()+1,vector<
34         <int> (b.size()+1,0));
35     vector<vector<pair<int,int>>> pre(a.size()
36         +1,vector<pair<int,int>> (b.size()
37         +1));
38     for(int i=0;i<a.size();i++){
39         for(int j=0;j<b.size();j++){
40             if(a[i] == b[j]){
41                 dp[i+1][j+1] = dp[i][j] + 1;
42                 pre[i+1][j+1] = {i,j};
43             }
44             else if(dp[i+1][j] >= dp[i][j
45                 +1]){
46                 dp[i+1][j+1] = dp[i+1][j];
47                 pre[i+1][j+1] = {i+1,j};
48             }
49             else {
50                 dp[i+1][j+1] = dp[i][j+1];
51                 pre[i+1][j+1] = {i,j+1};
52             }
53         }
54     }
55     int index1 = a.size(), index2 = b.size()
56     ;
57     string ans;
58     while(index1>0&&index2>0){
59         if(pre[index1][index2] == make_pair(
60             index1-1,index2-1)){
61             ans+=a[index1-1];
62         }
63         pair<int,int> u = pre[index1][index2
64             ];
65         index1= u.first;
66         index2= u.second;
67     }
68     for(int i=ans.size()-1;i>=0;i--)cout<<
69         ans[i];
70     return 0;
71 }

```

2.2 DP on Graph

```

1 //G.Longest Path
2 vector<vector<int>> G;
3 vector<int> in;
4 int n, m;
5 cin >> n >> m;
6 G.assign(n+1, {});
7 in.assign(n+1, 0);
8 while (m--) {
9     int u, v;
10    cin >> u >> v;
11    G[u].emplace_back(v);
12    ++in[v];
13 }
14 int solve(int n) {
15     vector<int> DP(G.size(), 0);

```

```

16     vector<int> Q;
17     for (int u = 1; u <= n; ++u)
18         if (in[u] == 0)
19             Q.emplace_back(u);
20     for (size_t i = 0; i < Q.size(); ++i) {
21         int u = Q[i];
22         for (auto v : G[u]) {
23             DP[v] = max(DP[v], DP[u] + 1);
24             if (--in[v] == 0)
25                 Q.emplace_back(v);
26         }
27     }
28     return *max_element(DP.begin(), DP.end());
29 }
30 //max_independent_set on tree
31 vector<int> DP[2];
32 int dfs(int u, int pick, int parent = -1) {
33     if (u == parent) return 0;
34     if (DP[pick][u]) return DP[pick][u];
35     if (Tree[u].size() == 1) return pick; //
36     //葉子
37     for (auto v : Tree[u]) {
38         if (pick == 0) {
39             DP[pick][u] += max(dfs(v, 0, u), dfs(v
40                 , 1, u));
41         }
42         else {
43             DP[pick][u] += dfs(v, 0, u);
44         }
45     }
46     return DP[pick][u] += pick;
47 }
48 int solve(int n) {
49     DP[0] = DP[1] = vector<int>(n+1, 0);
50     return max(dfs(1, 0), dfs(1, 1));
51 }
52 //Traveling Salesman // AtCoder
53 const int INF = 1e9;
54 int cost(vector<tuple<int,int,int>> &point,
55     int from, int to) {
56     auto [x,y,z] = point[from];
57     auto [X,Y,Z] = point[to];
58     return abs(X-x)+abs(Y-y)+max(0,Z-z);
59 }
60 //從一個點走到另一個點的花費
61 signed main() {
62     int n;cin>>n;
63     vector<tuple<int,int,int>> point(n);
64     for(auto &[x,y,z]:point) {
65         cin>>x>>y>>z;
66     }
67     vector<vector<int>> dp(1<<n,vector<int>
68         (n,INF));
69     //1<<n(2^n)代表1~n的所有子集，代表走過的
70     //點
71     //n代表走到的最後一個點
72     dp[0][0] = 0;
73     for(int i=1;i<(1<<n);i++){
74         for(int j=0;j<n;j++){
75             if(i & (1<<j)) {
76                 //j是走到的最後一個點，必須
77                 //要在i裡面
78                 for(int k=0;k<n;k++){
79                     dp[i][j] = min(dp[i][j],
80                         dp[i-(1<<j)][k]+cost
81                         (point,k,j));
82                 }
83             }
84         }
85     }
86     cout<<dp[(1<<n)-1][0]; //每個都要走到，要
87     //走回1
88     return 0;
89 }

```

2.3 is planar

```

1 struct FringeOpposedSubset {
2     deque<int> left, right;
3     FringeOpposedSubset() = default;
4     FringeOpposedSubset(int h) : left{h},
5         right() {}
6 };
7 template<typename T>
8 void extend(T& a, T& b, bool rev = false) {
9     rev ? a.insert(a.begin(), b.rbegin(), b.
10         rend())
11         : a.insert(a.end(), b.begin(), b.end()
12             );
13 }
14 struct Fringe {
15     deque<FringeOpposedSubset> FOPs;
16     Fringe(int h) : FOPs{{h}} {}
17     bool operator<(const Fringe& o) const {
18         return std::tie(FOPs.back().left.back(),
19             FOPs.front().left.front()) <
20             std::tie(o.FOPs.back().left.back(),
21                 o.FOPs.front().left.front());
22     }
23     void merge(Fringe& o) {
24         o.merge_t_alike_edges();
25         merge_t_opposite_edges_into(o);
26         if (FOPs.front().right.empty())
27             o.align_duplicates(FOPs.back().left.
28                 front());
29     }
30     else {
31         make_union_structure(o);
32         if (o.FOPs.front().left.size()) FOPs.
33             push_front(o.FOPs.front());
34     }
35     void merge_t_alike_edges() {
36         FringeOpposedSubset ans;
37         for (auto& FOP : FOPs) {
38             if (!FOP.right.empty()) throw
39                 runtime_error("Exception");
40             extend(ans.left, FOP.left);
41         }
42         FOPs = {ans};
43     }
44     void merge_t_opposite_edges_into(Fringe& o
45         ) {
46         while (FOPs.front().right.empty() &&
47             FOPs.back().left.empty() &&
48             FOPs.size() > 1) {
49             auto& f = FOPs.front();
50             auto& b = FOPs.back();
51             f.merge(b);
52             FOPs.pop_front();
53         }
54     }
55 };

```

```

37 FOPs.front().left.front() > o.
    FOPs.front().left.back()) {
38     extend(o.FOPs.front().right, FOPs.
        front().left);
39     FOPs.pop_front();
40 }
41 }
42 void align_duplicates(int dfs_h) {
43     if (FOPs.front().left.back() == dfs_h) {
44         FOPs.front().left.pop_back();
45         swap_side();
46     }
47 }
48 void swap_side() {
49     if (FOPs.front().left.empty() ||
50         (!FOPs.front().right.empty() &&
51         FOPs.front().left.back() > FOPs.
52         front().right.back())) {
53         swap(FOPs.front().left, FOPs.front().
54         right);
55     }
56 }
57 void make_onion_structure(Fringe& o) {
58     auto low = &FOPs.front().left, high = &
59     FOPs.front().right;
60     if (FOPs.front().left.front() >= FOPs.
61     front().right.front())
62         swap(low, high);
63     if (o.FOPs.front().left.back() < low->
64     front())
65         throw runtime_error("Exception");
66     if (o.FOPs.front().left.back() < high->
67     front()) {
68         extend(*low, o.FOPs.front().left, true
69         );
70         extend(*high, o.FOPs.front().right,
71         true);
72         o.FOPs.front().left.clear();
73         o.FOPs.front().right.clear();
74     }
75 }
76 auto lr_condition(int deep) const {
77     bool L = !FOPs.front().left.empty() &&
78     FOPs.front().left.front() >= deep;
79     bool R = !FOPs.front().right.empty() &&
80     FOPs.front().right.front() >= deep;
81     return make_pair(L, R);
82 }
83 void prune(int deep) {
84     auto [left, right] = lr_condition(deep);
85     while (!FOPs.empty() && (left || right))
86     {
87         if (left) FOPs.front().left.pop_front
88         ();
89         if (right) FOPs.front().right.
90         pop_front();
91         if (FOPs.front().left.empty() && FOPs.
92         front().right.empty())
93             FOPs.pop_front();
94         else swap_side();
95         if (!FOPs.empty()) tie(left, right) =
96         lr_condition(deep);
97     }
98 }
99 };
100
101 unique_ptr<Fringe> get_merged_fringe(deque<
102 unique_ptr<Fringe>>& upper) {
103     if (upper.empty()) return nullptr;
104     sort(upper.begin(), upper.end(), [](auto&
105     a, auto& b) { return *a < *b; });
106     for (auto it = next(upper.begin()); it !=
107     upper.end(); ++it)
108         upper.front()->merge(*it);
109     return move(upper.front());
110 }
111 void merge_fringes(vector<deque<unique_ptr<
112 Fringe>>>& fringes, int deep) {
113     auto mf = get_merged_fringe(fringes.back()
114 );
115     fringes.pop_back();
116     if (mf) {
117         mf->prune(deep);
118         if (mf->FOPs.size()) fringes.back().
119         push_back(move(mf));
120     }
121 }
122 struct Edge {
123     int from, to;
124     Edge(int from, int to) : from(from), to(to)
125     {}
126     bool operator==(const Edge& o) const {
127         return from == o.from && to == o.to;
128     }
129 };
130 struct Graph {
131     int n = 0;
132     vector<vector<int>> neighbor;
133     vector<Edge> edges;
134     void add_edge(int from, int to) {
135         if (from == to) return;
136         edges.emplace_back(from, to);
137         edges.emplace_back(to, from);
138     }
139     void build() {
140         sort(edges.begin(), edges.end(), [](
141         const auto& a, const auto& b) {
142             return a.from < b.from || (a.from == b
143             .from && a.to < b.to);
144         });
145         edges.erase(unique(edges.begin(), edges.
146         end()), edges.end());
147         n = 0;
148         for (auto& e : edges) n = max(n, max(e.
149         from, e.to) + 1);
150         neighbor.resize(n);
151         for (auto& e : edges) neighbor[e.from].
152         push_back(e.to);
153     }
154 };
155 Graph g;
156 vector<int> Deep;
157 vector<deque<unique_ptr<Fringe>>> fringes;
158 bool dfs(int x, int parent = -1) {
159     for (int y : g.neighbor[x]) {
160         if (y == parent) continue;
161         if (Deep[y] < 0) { // tree edge
162             fringes.push_back({});
163             Deep[y] = Deep[x] + 1;
164             if (!dfs(y, x)) return false;
165         } else if (Deep[x] > Deep[y]) { //
166         back edge
167     }
168 }
169
170 fringes.back().push_back(make_unique<
171 Fringe>(Deep[y]));
172 }
173 try {
174     if (fringes.size() > 1) merge_fringes(
175     fringes, Deep[parent]);
176 } catch (const exception& e) {
177     return false;
178 }
179 return true;
180 }
181 bool is_planar() {
182     Deep.assign(g.n, -1);
183     for (int i = 0; i < g.n; ++i) {
184         if (Deep[i] >= 0) continue;
185         fringes.clear();
186         Deep[i] = 0;
187         if (!dfs(i)) return false;
188     }
189     return true;
190 }
191 int main() {
192     int n, m, u, v;
193     cin >> n >> m;
194     for (int i = 0; i < m; ++i) {
195         cin >> u >> v;
196         g.add_edge(u, v);
197     }
198     g.build();
199     cout << (is_planar() ? "YES" : "NO") <<
200     endl;
201     return 0;
202 }
203
204 template<bool MAX>
205 struct line_container : std::multiset<
206 line_container_internal::line_t, std::
207 less<>> {
208     static const long long INF = std::
209     numeric_limits<long long>::max();
210     bool isect(iterator x, iterator y) {
211         if (y == end()) {
212             x->p = INF;
213             return 0;
214         }
215         if (x->k == y->k) {
216             x->p = (x->m > y->m ? INF : -INF);
217         } else {
218             x->p = floor_div(y->m - x->m, x->k - y
219             ->k);
220             return x->p >= y->p;
221         }
222     }
223     void add_line(long long k, long long m) {
224         if (!MAX) {
225             k = -k;
226             m = -m;
227         }
228         auto z = insert({k, m, 0}), y = z++, x =
229         y;
230         while (isect(y, z)) {
231             z = erase(z);
232         }
233         if (x != begin() && isect(--x, y)) {
234             isect(x, y = erase(y));
235         }
236         while ((y = x) != begin() && (--x)->p >=
237         y->p) {
238             isect(x, erase(y));
239         }
240     }
241     long long get(long long x) {
242         assert(!empty());
243         auto l = *lower_bound(x);
244         return (l.k * x + l.m) * (MAX ? +1 : -1)
245         ;
246     }
247 };
248
249 // Usually used for DP 斜率優化
250 template<class T>
251 T floor_div(T a, T b) {
252     return a / b - ((a ^ b) < 0 && a % b != 0)
253     ;
254 }
255 template<class T>
256 T ceil_div(T a, T b) {
257     return a / b + ((a ^ b) > 0 && a % b != 0)
258     ;
259 }
260 namespace line_container_internal {
261 struct line_t {
262     mutable long long k, m, p;
263     inline bool operator<(const line_t& o)
264     const { return k < o.k; }
265     inline bool operator<(long long x) const {
266         return p < x; }
267 };
268 void compute(int L, int R, int optL, int
269 optR) {
270     if (L > R)
271         return;
272     int mid = L + (R - L) / 2;
273     DP[mid] = INF;
274     int opt = -1;
275     for (int k = optL; k <= min(mid - 1, optR)
276     ; k++) {
277         if (DP[mid] > f(k) + w(k, mid)) {
278             DP[mid] = f(k) + w(k, mid);
279             opt = k;
280         }
281     }
282 }

```

2.4 LineContainer

2.5 整體二分


```

12 }
13 compute(L, mid - 1, optL, opt);
14 compute(mid + 1, R, opt, optR);
15 }
16 // compute(1, n, 0, n);

```

2.6 斜率優化-動態凸包

```

1 struct Line
2 {
3     mutable ll a, b, l;
4     Line(ll _a, ll _b, ll _l) : a(_a), b(_b)
5         , l(_l) {}
6     bool operator<(const Line &rhs) const
7     {
8         return make_pair(-a, -b) < make_pair
9             (-rhs.a, -rhs.b);
10    }
11    bool operator<(ll rhs_l) const
12    {
13        return l < rhs_l;
14    }
15 };
16 struct ConvexHullMin : std::multiset<Line,
17     std::less<>>
18 {
19     static const ll INF = (1ll << 60);
20     static ll DivCeil(ll a, ll b)
21     {
22         return a / b - ((a ^ b) < 0 && a % b
23             );
24     }
25     bool Intersect(iterator x, iterator y)
26     {
27         if (y == end())
28         {
29             x->l = INF;
30             return false;
31         }
32         if (x->a == y->a)
33         {
34             x->l = x->b < y->b ? INF : -INF;
35         }
36         else
37         {
38             x->l = DivCeil(y->b - x->b, x->a
39                 - y->a);
40         }
41         return x->l >= y->l;
42     }
43     void Insert(ll a, ll b)
44     {
45         auto z = insert(Line(a, b, 0)), y =
46             z++, x = y;
47         while (Intersect(y, z))
48             z = erase(z);
49         if (x != begin() && Intersect(--x, y))
50             Intersect(x, y = erase(y));
51         while ((y = x) != begin() && (--x)->
52             l >= y->l)
53             Intersect(x, erase(y));

```

```

48 }
49 ll query(ll x) const
50 {
51     auto l = *lower_bound(x);
52     return l.a * x + l.b;
53 }
54 } convexhull;
55
56 const ll maxn = 200005;
57 ll s[maxn];
58 ll f[maxn];
59 ll dp[maxn];
60 // CSES monster game2
61 int main()
62 {
63     Crbubble
64     ll n, m, i, k, t;
65     cin >> n >> f[0];
66     for(i=1; i<=n; i++) cin >> s[i];
67     for(i=1; i<=n; i++) cin >> f[i];
68     convexhull.Insert(f[0], 0);
69     for(i=1; i<=n; i++)
70     {
71         dp[i] = convexhull.query(s[i]);
72         convexhull.Insert(f[i], dp[i]);
73     }
74     cout << dp[n] << endl;
75     return 0;

```

2.7 斜率優化-單調隊列

```

1 struct line {
2     ll a, b;
3     line(ll _a, ll _b): a(_a), b(_b) {}
4     ll operator()(ll x) {
5         return a * x + b;
6     }
7 };
8
9 bool remove(line &L1, line &L2, line &now)
10 {
11     // L1 + now remove L2 ?
12     // p1_now = (now.b-L1.b)/(L1.a-now.a);
13     // p2_now = (now.b-L2.b)/(L2.a-now.a);
14     // return p1 >= p2
15     return (now.b-L1.b)*(L2.a-now.a) >= (now
16         .b-L2.b)*(L1.a-now.a);
17 }
18
19 const ll maxn = 200005;
20 ll s[maxn];
21 ll f[maxn];
22 ll dp[maxn];
23 // 斜率優化-單調對列
24 // Monster Game I
25 // https://cses.fi/problemset/task/2084/
26 // 斜率單調、查詢單調
27
28 int main() {
29     ll n, m, i, k, t;
30     cin >> n >> m;
31     for(i=1; i<=n; i++) cin >> s[i];
32     for(i=1; i<=n; i++) cin >> f[i];

```

```

31 deque<line> q;
32 q.push_back(line(m, 0));
33 for(i=1; i<=n; i++) {
34     while(q.size() >= 2 && q[0](s[i]) >=
35         q[1](s[i]))
36         q.pop_front();
37     dp[i] = q[0](s[i]);
38     line now = line(f[i], dp[i]);
39     while(q.size() >= 2 && remove(q[q.
40         size()-2], q[q.size()-1], now))
41         q.pop_back();
42     q.push_back(now);
43 }
44 cout << dp[n] << endl;
45 return 0;

```

3 Data Structure

3.1 2D BIT

```

1 //2維BIT
2 #define lowbit(x) (x&-x)
3
4 class BIT {
5     int n;
6     vector<int> bit;
7
8 public:
9     void init(int _n) {
10         n = _n;
11         bit.resize(n + 1);
12         for(auto &b : bit) b = 0;
13     }
14     int query(int x) const {
15         int sum = 0;
16         for(; x; x -= lowbit(x))
17             sum += bit[x];
18         return sum;
19     }
20     void modify(int x, int val) {
21         for(; x <= n; x += lowbit(x))
22             bit[x] += val;
23     }
24 };
25
26 class BIT2D {
27     int m;
28     vector<BIT> bit1D;
29
30 public:
31     void init(int _m, int _n) {
32         m = _m;
33         bit1D.resize(m + 1);
34         for(auto &b : bit1D) b.init(_n);
35     }
36     int query(int x, int y) const {
37         int sum = 0;
38         for(; x; x -= lowbit(x))
39             sum += bit1D[x].query(y);
40     }

```

```

41     return sum;
42 }
43 void modify(int x, int y, int val) {
44     for(; x <= m; x += lowbit(x))
45         bit1D[x].modify(y, val);
46 }
47 };

```

3.2 BinaryTrie

```

1 template<class T>
2 struct binary_trie {
3     public:
4     binary_trie() {
5         new_node();
6     }
7
8     void clear() {
9         trie.clear();
10        new_node();
11    }
12
13    void insert(T x) {
14        for(int i = B - 1, p = 0; i >= 0; i--) {
15            int y = x >> i & 1;
16            if(trie[p].go[y] == 0) {
17                trie[p].go[y] = new_node();
18            }
19            p = trie[p].go[y];
20            trie[p].cnt += 1;
21        }
22    }
23
24    void erase(T x) {
25        for(int i = B - 1, p = 0; i >= 0; i--) {
26            p = trie[p].go[x >> i & 1];
27            trie[p].cnt -= 1;
28        }
29    }
30
31    bool contains(T x) {
32        for(int i = B - 1, p = 0; i >= 0; i--) {
33            p = trie[p].go[x >> i & 1];
34            if(trie[p].cnt == 0) {
35                return false;
36            }
37        }
38        return true;
39    }
40
41    T get_min() {
42        return get_xor_min(0);
43    }
44
45    T get_max() {
46        return get_xor_max(0);
47    }
48
49    T get_xor_min(T x) {
50        T ans = 0;
51        for(int i = B - 1, p = 0; i >= 0; i--) {
52            int y = x >> i & 1;
53            int z = trie[p].go[y];

```

```

54     if(z > 0 && trie[z].cnt > 0) {
55         p = z;
56     } else {
57         ans |= T(1) << i;
58         p = trie[p].go[y ^ 1];
59     }
60 }
61 return ans;
62 }
63
64 T get_xor_max(T x) {
65     T ans = 0;
66     for(int i = B - 1, p = 0; i >= 0; i--) {
67         int y = x >> i & 1;
68         int z = trie[p].go[y ^ 1];
69         if(z > 0 && trie[z].cnt > 0) {
70             ans |= T(1) << i;
71             p = z;
72         } else {
73             p = trie[p].go[y];
74         }
75     }
76     return ans;
77 }
78
79 private:
80     static constexpr int B = sizeof(T) * 8;
81
82     struct Node {
83         std::array<int, 2> go = {};
84         int cnt = 0;
85     };
86
87     std::vector<Node> trie;
88
89     int new_node() {
90         trie.emplace_back();
91         return (int) trie.size() - 1;
92     }
93 };

```

3.3 BIT

```

1 #define lowbit(x) x & -x
2
3 void modify(vector<int> &bit, int idx, int
4     val) {
5     for(int i = idx; i <= bit.size(); i+=
6         lowbit(i)) bit[i] += val;
7 }
8
9 int query(vector<int> &bit, int idx) {
10     int ans = 0;
11     for(int i = idx; i > 0; i-= lowbit(i)) ans
12         += bit[i];
13     return ans;
14 }
15
16 // the first i s.t. a[1]+...+a[i] >= k
17 int findK(vector<int> &bit, int k) {
18     int idx = 0, res = 0;
19     int mx = __lg(bit.size()) + 1;
20     for(int i = mx; i >= 0; i--) {

```

```

18     if((idx | (1<<i)) > bit.size()) continue
19     ;
20     if(res + bit[idx | (1<<i)] < k) {
21         idx = (idx | (1<<i));
22         res += bit[idx];
23     }
24     return idx + 1;
25 }
26 //O(n)建bit
27 for (int i = 1; i <= n; ++i) {
28     bit[i] += a[i];
29     int j = i + lowbit(i);
30     if (j <= n) bit[j] += bit[i];
31 }

```

3.4 DSU

```

1 struct DSU {
2     vector<int> dsu, sz;
3     DSU(int n) {
4         dsu.resize(n + 1);
5         sz.resize(n + 1, 1);
6         for (int i = 0; i <= n; i++) dsu[i] = i;
7     }
8     int find(int x) {
9         return (dsu[x] == x ? x : dsu[x] = find(
10             dsu[x]));
11     }
12     int unite(int a, int b) {
13         a = find(a), b = find(b);
14         if(a == b) return 0;
15         if(sz[a] > sz[b]) swap(a, b);
16         dsu[a] = b;
17         sz[b] += sz[a];
18         return 1;
19     }
20 };

```

3.5 DynamicMST

```

1 int cnt[maxn], cost[maxn], st[maxn], ed[maxn]
2 ;
3 pair<int, int> qr[maxn];
4 // qr[i].first = id of edge to be changed,
5 // qr[i].second = weight after operation
6 // cnt[i] = number of operation on edge i
7 // call solve(0, q - 1, v, 0), where v
8 // contains edges i such that cnt[i] == 0
9
10 void contract(int l, int r, vector<int> v,
11     vector<int> &x, vector<int> &y) {
12     sort(v.begin(), v.end(), [&](int i, int j)
13         {
14             if (cost[i] == cost[j]) return i < j;
15             return cost[i] < cost[j];
16         });
17     djs.save();
18     for (int i = l; i <= r; ++i) djs.merge(st[
19         qr[i].first], ed[qr[i].first]);

```

```

14 for (int i = 0; i < (int)v.size(); ++i) {
15     if (djs.find(st[v[i]]) != djs.find(ed[v[
16         i]])) {
17         x.push_back(v[i]);
18         djs.merge(st[v[i]], ed[v[i]]);
19     }
20 }
21 djs.undo();
22 djs.save();
23 for (int i = 0; i < (int)x.size(); ++i)
24     djs.merge(st[x[i]], ed[x[i]]);
25
26 for (int i = 0; i < (int)v.size(); ++i) {
27     if (djs.find(st[v[i]]) != djs.find(ed[v[
28         i]])) {
29         y.push_back(v[i]);
30         djs.merge(st[v[i]], ed[v[i]]);
31     }
32 }
33 djs.undo();
34
35 void solve(int l, int r, vector<int> v, long
36     long c) {
37     if (l == r) {
38         cost[qr[l].first] = qr[l].second;
39         if (st[qr[l].first] == ed[qr[l].first])
40             {
41                 printf("%lld\n", c);
42                 return;
43             }
44         int minv = qr[l].second;
45         for (int i = 0; i < (int)v.size(); ++i)
46             minv = min(minv, cost[v[i]]);
47         printf("%lld\n", c + minv);
48         return;
49     }
50     int m = (l + r) >> 1;
51     vector<int> lv = v, rv = v;
52     vector<int> x, y;
53     for (int i = m + 1; i <= r; ++i) {
54         cnt[qr[i].first]--;
55         if (cnt[qr[i].first] == 0) lv.push_back(
56             qr[i].first);
57     }
58     contract(l, m, lv, x, y);
59     long long lc = c, rc = c;
60     djs.save();
61     for (int i = 0; i < (int)x.size(); ++i) {
62         lc += cost[x[i]];
63         djs.merge(st[x[i]], ed[x[i]]);
64     }
65     solve(l, m, y, lc);
66     djs.undo();
67     x.clear(), y.clear();
68     for (int i = m + 1; i <= r; ++i) cnt[qr[i]
69         ].first++;
70     for (int i = l; i <= m; ++i) {
71         cnt[qr[i].first]--;
72         if (cnt[qr[i].first] == 0) rv.push_back(
73             qr[i].first);
74     }
75     contract(m + 1, r, rv, x, y);
76     djs.save();
77     for (int i = 0; i < (int)x.size(); ++i) {
78         rc += cost[x[i]];
79         djs.merge(st[x[i]], ed[x[i]]);

```

```

71 }
72 solve(m + 1, r, y, rc);
73 djs.undo();
74 for (int i = l; i <= m; ++i) cnt[qr[i].
75     first]++;

```

3.6 Dynamic Segment Tree

```

1 using ll = long long;
2 struct node {
3     node *l, *r; ll sum;
4     void pull() {
5         sum = 0;
6         for(auto x : {l, r}) if(x) sum += x->sum;
7     }
8     node(int v = 0): sum(v) {l = r = nullptr;}
9 };
10
11 void upd(node* o, int x, ll v, int l, int r
12     ) {
13     if(!o) o = new node;
14     if(l == r) return o->sum += v, void();
15     int m = (l + r) / 2;
16     if(x <= m) upd(o->l, x, v, l, m);
17     else upd(o->r, x, v, m+1, r);
18     o->pull();
19 }
20
21 ll qry(node* o, int ql, int qr, int l, int r
22     ) {
23     if(!o) return 0;
24     if(ql <= l && r <= qr) return o->sum;
25     int m = (l + r) / 2; ll ret = 0;
26     if(ql <= m) ret += qry(o->l, ql, qr, l, m);
27     if(qr > m) ret += qry(o->r, ql, qr, m+1, r);
28     return ret;

```

3.7 Kruskal

```

1 vector<tuple<int,int,int>> Edges;
2 int kruskal(int N) {
3     int cost = 0;
4     sort(Edges.begin(), Edges.end());
5
6     DisjointSet ds(N);
7
8     sort(Edges.begin(), Edges.end());
9     for(auto [w, s, t] : Edges) {
10         if (!ds.same(s, t)) {
11             cost += w;
12             ds.unit(s, t);
13         }
14     }
15     return cost;
16 }

```

3.8 Lazytag Segment Tree

```

1 using ll = long long;
2 const int N = 2e5 + 5;
3 #define lc(x) (x << 1)
4 #define rc(x) (x << 1 | 1)
5
6 // [1,n]
7 // tag[i] represents the modifications to be
8 // applied to the children,
9 // while seg[i] has already been modified.
10 ll seg[N << 2], tag[N << 2];
11 int n;
12
13 void pull(int id) {
14     seg[id] = seg[lc(id)] + seg[rc(id)];
15 }
16
17 void push(int id, int l, int r) {
18     if (tag[id]) {
19         int m = (l + r) >> 1;
20         tag[lc(id)] += tag[id], tag[rc(id)] +=
21             tag[id];
22         seg[lc(id)] += (m - l + 1) * tag[id],
23             seg[rc(id)] += (r - m) * tag[id];
24         tag[id] = 0;
25     }
26
27 void upd(int ql, int qr, ll v, int l = 1,
28     int r = n, int id = 1) {
29     if (ql <= l && r <= qr) return tag[id] +=
30         v, seg[id] += (r - l + 1) * v, void();
31     push(id, l, r);
32     int m = (l + r) >> 1;
33     if (ql <= m) upd(ql, qr, v, l, m, lc(id));
34     if (qr > m) upd(ql, qr, v, m + 1, r, rc(id));
35     pull(id);
36 }
37
38 ll qry(int ql, int qr, int l = 1, int r = n,
39     int id = 1) {
40     if (ql <= l && r <= qr) return seg[id];
41     push(id, l, r);
42     int m = (l + r) >> 1; ll ret = 0;
43     if (ql <= m) ret += qry(ql, qr, l, m, lc(id));
44     if (qr > m) ret += qry(ql, qr, m + 1, r, rc(id));
45     return ret;
46 }

```

3.9 LiChaoST

```

1 struct L {
2     ll m, k, id;
3     L() : id(-1) {}
4     L(ll a, ll b, ll c) : m(a), k(b), id(c) {}
5     ll at(ll x) { return m * x + k; }
6 };
7 class LiChao { // maintain max

```

```

8 private:
9     int n; vector<L> nodes;
10     void insert(int l, int r, int rt, L ln) {
11         int m = (l + r) >> 1;
12         if (nodes[rt].id == -1)
13             return nodes[rt] = ln, void();
14         bool atLeft = nodes[rt].at(1) < ln.at(1)
15             ;
16         if (nodes[rt].at(m) < ln.at(m))
17             atLeft ^= 1, swap(nodes[rt], ln);
18         if (r - l == 1) return;
19         if (atLeft) insert(l, m, rt << 1, ln);
20         else insert(m, r, rt << 1 | 1, ln);
21     }
22     ll query(int l, int r, int rt, ll x) {
23         int m = (l + r) >> 1; ll ret = -INF;
24         if (nodes[rt].id != -1) ret = nodes[rt].
25             at(x);
26         if (r - l == 1) return ret;
27         if (x < m) return max(ret, query(l, m,
28             rt << 1, x));
29         return max(ret, query(m, r, rt << 1 | 1,
30             x));
31     }
32 public:
33     LiChao(int n_) : n(n_), nodes(n * 4) {}
34     void insert(L ln) { insert(0, n, 1, ln); }
35     ll query(ll x) { return query(0, n, 1, x); }
36 };

```

3.10 pbds

```

1 #include <ext/pb_ds/tree_policy.hpp>
2 #include <ext/pb_ds/assoc_container.hpp>
3 using namespace __gnu_pbds;
4
5 template <class T>
6 using ordered_set = tree<T, null_type, less<
7     T>, rb_tree_tag,
8     tree_order_statistics_node_update>;
9
10 template <class T>
11 // ordered_multiset: do not use erase method
12 // , use myerase() instead
13 using ordered_multiset = tree<T, null_type,
14     less_equal<T>, rb_tree_tag,
15     tree_order_statistics_node_update>;
16
17 template <class T>
18 void myerase(ordered_multiset<T> &ss, T v)
19 {
20     T rank = ss.order_of_key(v); //
21     // Number of elements that are less
22     // than v in ss
23     auto it = ss.find_by_order(rank); //
24     // Iterator that points to the element
25     // which index = rank
26     ss.erase(it);
27 }

```

3.11 Persistent DSU

```

1 int rk[200001] = {};
2 struct Persistent_DSU{
3     rope<int>*p;
4     int n;
5     Persistent_DSU(int _n = 0):n(_n){
6         if(n==0)return;
7         p = new rope<int>;
8         int tmp[n+1] = {};
9         for(int i = 1;i<=n;++i)tmp[i] = i;
10        p->append(tmp,n+1);
11    }
12    Persistent_DSU(const Persistent_DSU &tmp){
13        p = new rope<int>(*tmp.p);
14        n = tmp.n;
15    }
16    int Find(int x){
17        int px = p->at(x);
18        return px==x?x:Find(px);
19    }
20    bool Union(int a,int b){
21        int pa = Find(a),pb = Find(b);
22        if(pa==pb)return 0;
23        if(rk[pa]<rk[pb])swap(pa,pb);
24        p->replace(pb,pa);
25        if(rk[pa]==rk[pb])rk[pa]++;
26        return 1;
27    }
28 };

```

3.12 Persistent Segment Tree

```

1 using ll = long long;
2 int n;
3
4 struct node {
5     node *l, *r; ll sum;
6     void pull() {
7         sum = 0;
8         for (auto x : {l, r})
9             if (x) sum += x->sum;
10    }
11    node(int v = 0): sum(v) {l = r = nullptr;}
12    *root = nullptr;
13
14 void upd(node *prv, node* cur, int x, int v,
15     int l = 1, int r = n) {
16     if (l == r) return cur->sum = v, void();
17     int m = (l + r) >> 1;
18     if (x <= m) cur->r = prv->r, upd(prv->l,
19         cur->l = new node(x, v, l, m));
20     else cur->l = prv->l, upd(prv->r, cur->r =
21         new node(x, v, m + 1, r));
22     cur->pull();
23 }
24
25 ll qry(node* a, node* b, int ql, int qr, int
26     l = 1, int r = n) {
27     if (ql <= l && r <= qr) return b->sum - a
28         ->sum;
29     int m = (l + r) >> 1; ll ret = 0;

```

```

25 if (ql <= m) ret += qry(a->l, b->l, ql, qr,
26     l, m);
27 if (qr > m) ret += qry(a->r, b->r, ql, qr,
28     m + 1, r);
29 return ret;
30 }

```

3.13 Prim

```

1 int cost[MAX_V][MAX_V]; //Edge的權重 (不存在
2     時為INF)
3 int mincost[MAX_V]; //來自集合X的邊的最小權重
4 bool used[MAX_V]; //頂點i是否包含在X之中
5 int V; //頂點數
6
7 int prim() {
8     for(int i = 0; i < V; i++) {
9         mincost[i] = INF;
10        used[i] = false;
11    }
12    mincost[0] = 0;
13    int res = 0;
14    while(true) {
15        int v = -1;
16        //從不屬於X的頂點中尋找會讓來自X的邊
17        //之權重最小的頂點
18        for(int u = 0; u < V; u++) {
19            if(!used[u] && (v == -1 || mincost
20                [u] < mincost[v])) v = u;
21        }
22        if(v == -1) break;
23        used[v] = true; //將頂點v追加至X
24        res += mincost[v]; //加上邊的權重
25        for(int u = 0; u < V; u++) {
26            mincost[u] = min(mincost[u], cost
27                [v][u]);
28        }
29    }
30    return res;
31 }

```

3.14 SegmentTree

```

1 //build
2 const int N = 100000 + 9;
3 int a[N]; //葉
4 int seg[4 * N];
5 void build(int id, int l, int r) { // 編號為
6     // id 的節點 · 存的區間為 [l, r]
7     if (l == r) {
8         seg[id] = a[l]; // 葉節點的值
9         return;
10    }
11    int mid = (l + r) / 2; // 將區間切成兩半
12    build(id * 2, l, mid); // 左子節點
13    build(id * 2 + 1, mid + 1, r); // 右子節
14    點

```

```

14     seg[id] = seg[id * 2] + seg[id * 2 + 1]
15 }
16 //區間查詢
17
18 int query(int id, int l, int r, int ql, int
19 qr) {
20     if (r < ql || qr < l) return 0; //若目前
    的區間與詢問的區間的交集為空的話，
    return 0
21     if (ql <= l && r <= qr) return seg[id];
    //若目前的區間是詢問的區間的子集的
    話，則終止，並回傳當前節點的答案
22     int mid = (l + r) / 2;
23     return query(id * 2, l, mid, ql, qr) //
    左
24     + query(id * 2 + 1, mid + 1, r, ql,
    qr); //右
25     //否則，往左、右進行遞迴
26 }
27
28 //單點修改
29
30 void modify(int id, int l, int r, int i, int
31 x) {
32     if (l == r) {
33         seg[id] = x; // 將a[i]改成x
34         //seg[id] += x; // 將a[i]加上x
35         return;
36     }
37     int mid = (l + r) / 2;
38     // 根據修改的點在哪裡，來決定要往哪個子
    樹進行DFS
39     if (i <= mid) modify(id * 2, l, mid, i,
    x); //左
40     else modify(id * 2 + 1, mid + 1, r, i, x
    ); //右
41     seg[id] = seg[id * 2] + seg[id * 2 + 1];
42 }

```

3.15 sparse table

```

1 //CSES Static Range Minimum Queries
2 #define inf 1e9
3 vector<vector<int>> st;
4
5 void build_sparse_table(int n) {
6     st.assign(__lg(n)+1, vector<int> (n+1, inf))
7     ;
8     for(int i=1; i<=n; i++) cin>>st[0][i];
9     for(int i=1; (1<<i)<=n; i++) {
10         for(int j=1; j + (1<<i) - 1 <= n; j++) {
11             st[i][j] = min(st[i-1][j], st[i-1][j
12                 +(1<<(i-1))]);
13         }
14     }
15 }
16
17 int query(int l, int r) {

```

```

16     int k = __lg(r - l + 1);
17     return min(st[k][l], st[k][r-(1<<k)+1]);
18 }
19
20 signed main() {
21     int n, q; cin>>n>>q;
22     build_sparse_table(n);
23     while(q--) {
24         int l, r; cin>>l>>r;
25         cout<<query(l, r)<<'\n';
26     }
27 }

```

3.16 TimingSegmentTree

```

1 template<class T, class D> struct
    timing_segment_tree {
2     struct node {
3         int l, r;
4         vector<T> opt;
5     };
6     vector<node> arr;
7     void build(int l, int r, int idx = 1) {
8         if (idx == 1) arr.resize((r-l+1)<<2);
9         if (l == r) {
10             arr[idx].l = arr[idx].r = l;
11             arr[idx].opt.clear();
12             return;
13         }
14         int m = (l+r)>>1;
15         build(l, m, idx<<1);
16         build(m+1, r, idx<<1|1);
17         arr[idx].l = l, arr[idx].r = r;
18         arr[idx].opt.clear();
19     }
20     void update(int ql, int qr, T k, int idx = 1)
21     {
22         if (ql<=arr[idx].l and arr[idx].r<=qr) {
23             arr[idx].opt.push_back(k);
24             return;
25         }
26         int m = (arr[idx].l+arr[idx].r)>>1;
27         if (ql<=m) update(ql, qr, k, idx<<1);
28         if (qr>m) update(ql, qr, k, idx<<1|1);
29     }
30     void dfs(D &d, vector<int> &ans, int idx = 1)
31     {
32         int cnt = 0;
33         for (auto [a, b]: arr[idx].opt) {
34             if (d.Union(a, b)) cnt++;
35         }
36         if (arr[idx].l == arr[idx].r) ans[arr[idx].l
37             ] = d.comps;
38         else {
39             dfs(d, ans, idx<<1);
40             dfs(d, ans, idx<<1|1);
41         }
42         while (cnt-->0) d.undo();
43     }
44 };

```

3.17 回滾並查集

```

1 struct dsu_undo {
2     vector<int> sz, p;
3     int comps;
4     dsu_undo(int n) {
5         sz.assign(n+5, 1);
6         p.resize(n+5);
7         for (int i = 1; i<=n; ++i) p[i] = i;
8         comps = n;
9     }
10    vector<pair<int, int>> opt;
11    int Find(int x) {
12        return x==p[x]?x:Find(p[x]);
13    }
14    bool Union(int a, int b) {
15        int pa = Find(a), pb = Find(b);
16        if (pa==pb) return 0;
17        if (sz[pa]<sz[pb]) swap(pa, pb);
18        sz[pa]+=sz[pb];
19        p[pb] = pa;
20        opt.push_back({pa, pb});
21        comps--;
22        return 1;
23    }
24    void undo() {
25        auto [pa, pb] = opt.back();
26        opt.pop_back();
27        p[pb] = pb;
28        sz[pa]-=sz[pb];
29        comps++;
30    }
31 };

```

3.18 掃描線 + 線段樹

```

1 //CSES Area of Rectangle
2 #define pb push_back
3 #define int long long
4 #define mid ((l + r) >> 1)
5 #define lc (p << 1)
6 #define rc ((p << 1) | 1)
7 struct ooo {
8     int x, l, r, v;
9 };
10 const int inf = 1e6;
11 array<int, 8000004> man, tag, cnt;
12 vector<ooo> Q;
13 bool cmp(ooo a, ooo b) {
14     return a.x < b.x;
15 }
16 void pull(int p) {
17     man[p] = min(man[lc], man[rc]);
18     if (man[lc] < man[rc]) cnt[p] = cnt[lc];
19     else if (man[rc] < man[lc]) cnt[p] = cnt[rc];
20     else cnt[p] = cnt[lc] + cnt[rc];
21 }
22 void push(int p) {
23     man[lc] += tag[p];
24     man[rc] += tag[p];
25     tag[lc] += tag[p];

```

```

26     tag[rc] += tag[p];
27     tag[p] = 0;
28 }
29 void build(int p, int l, int r) {
30     if (l == r) {
31         cnt[p] = 1;
32         return;
33     }
34     build(lc, l, mid);
35     build(rc, mid + 1, r);
36     pull(p);
37 }
38 void update(int p, int l, int r, int ql, int
39 qr, int x) {
40     if (ql > r || qr < l) return;
41     if (ql <= l && qr >= r) {
42         man[p] += x;
43         tag[p] += x;
44         return;
45     }
46     push(p);
47     update(lc, l, mid, ql, qr, x);
48     update(rc, mid + 1, r, ql, qr, x);
49     pull(p);
50 }
51 signed main() {
52     int n, x1, y1, x2, y2, p = 0, sum = 0;
53     cin >> n;
54     for (int i = 1; i <= n; ++i) {
55         cin >> x1 >> y1 >> x2 >> y2;
56         Q.pb({x1, y1, y2 - 1, 1});
57         Q.pb({x2, y1, y2 - 1, -1});
58     }
59     sort(Q.begin(), Q.end(), cmp);
60     build(1, -inf, inf);
61     for (int i = -inf; i < inf; i++) {
62         while (p < Q.size() && Q[p].x == i) {
63             auto [x, l, r, v] = Q[p++];
64             update(1, -inf, inf, l, r, v);
65         }
66         sum += 2 * inf + 1 - cnt[1];
67     }
68     cout << sum << "\n";
69 }
70 //長方形面積
71 long long AreaOfRectangles(vector<tuple<int,
72     int, int, int>> v) {
73     vector<tuple<int, int, int, int>> tmp;
74     int L = INT_MAX, R = INT_MIN;
75     for (auto [x1, y1, x2, y2]: v) {
76         tmp.push_back({x1, y1+1, y2, 1});
77         tmp.push_back({x2, y1+1, y2, -1});
78         R = max(R, y2);
79         L = min(L, y1);
80     }
81     vector<long long> seg((R-L+1)<<2), tag((R-L
82         +1)<<2);
83     sort(tmp.begin(), tmp.end());
84     function<void(int, int, int, int, int)>
85     update = [&](int ql, int qr, int val, int
86         l, int r, int idx) {
87         if (ql<=l and r<=qr) {
88             tag[idx] += val;
89             if (tag[idx]) seg[idx] = r-l+1;
90             else if (l==r) seg[idx] = 0;

```



```

86     else seg[idx] = seg[idx<<1]+seg[idx
      <<1|1];
87     return;
88 }
89 int m = (1+r)>>1;
90 if(q1<=m)update(q1,qr,val,l,m,idx<<1);
91 if(qr>m)update(q1,qr,val,m+1,r,idx<<1|1)
    ;
92 if(tag[idx])seg[idx] = r-l+1;
93 else seg[idx] = seg[idx<<1]+seg[idx
    <<1|1];
94 };
95 long long last_pos = 0,ans = 0;
96 for(auto [pos,l,r,val]:tmp){
97     ans+=(pos-last_pos)*seg[1];
98     update(1,r,val,L,R,1);
99     last_pos = pos;
100 }
101 return ans;
102 }
103 // CSES Intersection Points
104 #define int long long
105 #define pb push_back
106 struct line{
107     int p, l, r;
108 };
109 const int inf = 1e6 + 1;
110 array<int, 2000004> BIT;
111 vector<line> A, Q;
112 bool cmp(line a, line b){
113     return a.p < b.p;
114 }
115 void update(int p, int x){
116     for(; p < 2000004; p += p & -p) BIT[p]
        += x;
117 }
118 int query(int p){
119     int sum = 0;
120     for(; p; p -= p & -p) sum += BIT[p];
121     return sum;
122 }
123 int run(){
124     int ans = 0, p = 0;
125     for(auto [t, l, r] : Q){
126         while(p < A.size()){
127             auto [x, y, v] = A[p];
128             if(x > t) break;
129             update(y, v);
130             p++;
131         }
132         ans += query(r) - query(l - 1);
133     }
134     return ans;
135 }
136 signed main(){
137     int n, x1, x2, y1, y2;
138     cin >> n;
139     for(int i = 0; i < n; i++){
140         cin >> x1 >> y1 >> x2 >> y2;
141         x1 += inf, x2 += inf, y1 += inf, y2
            += inf;
142         if(x1 == x2) Q.pb({x1, y1, y2});
143         else A.pb({x1, y1, 1}), A.pb({x2 +
            1, y2, -1});
144     }
145 }

```

```

146     sort(Q.begin(), Q.end(), cmp);
147     sort(A.begin(), A.end(), cmp);
148     cout << run() << "\n";
149 }

```

3.19 陣列上 Treap

```

1 struct Treap {
2     Treap *lc = nullptr, *rc = nullptr;
3     unsigned pri, sz;
4     long long Val, Sum;
5     Treap(int Val):pri(rand()),sz(1),Val(Val),
        Sum(Val),Tag(false) {}
6     void pull();
7     bool Tag;
8     void push();
9 } *root;
10 inline unsigned sz(Treap *x) {
11     return x ? x->sz:0;
12 }
13 inline void Treap::push() {
14     if(!Tag) return ;
15     swap(lc,rc);
16     if(lc) lc->Tag ^= Tag;
17     if(rc) rc->Tag ^= Tag;
18     Tag = false;
19 }
20 inline void Treap::pull() {
21     sz = 1;
22     Sum = Val;
23     if(lc) {
24         sz += lc->sz;
25         Sum += lc->Sum;
26     }
27     if(rc) {
28         sz += rc->sz;
29         Sum += rc->Sum;
30     }
31 }
32 Treap *merge(Treap *a, Treap *b) {
33     if(!a || !b) return a ? a : b;
34     if(a->pri < b->pri) {
35         a->push();
36         a->rc = merge(a->rc,b);
37         a->pull();
38         return a;
39     }
40     else {
41         b->push();
42         b->lc = merge(a,b->lc);
43         b->pull();
44         return b;
45     }
46 }
47 pair<Treap *,Treap *> splitK(Treap *x,
    unsigned K) {

```

```

55     Treap *a = nullptr, *b = nullptr;
56     if(!x) return {a,b};
57     x->push();
58     unsigned leftSize = sz(x->lc) + 1;
59     if(K >= leftSize) {
60         a = x;
61         tie(a->rc,b) = splitK(x->rc, K -
            leftSize);
62     }
63     else {
64         b = x;
65         tie(a, b->lc) = splitK(x->lc, K);
66     }
67     x->pull();
68     return {a,b};
69 }
70 Treap *init(const vector<int> &a) {
71     Treap *root = nullptr;
72     for(size_t i = 0; i < a.size(); i++) {
73         root = merge(root,new Treap(a[i]));
74     }
75     return root;
76 }
77 long long query(Treap *root, unsigned ql,
    unsigned qr) {
78     auto [a,b] = splitK(root,ql);
79     auto [c,d] = splitK(b,qr-ql+1);
80     c->push();
81     long long Sum = c->Sum;
82     root = merge(a,merge(c,d));
83     return Sum;
84 }
85 void Reverse(Treap *root, unsigned ql,
    unsigned qr) {
86     auto [a,b] = splitK(root,ql);
87     auto [c,d] = splitK(b,qr-ql+1);
88     c->Tag ^= true;
89     root = merge(a, merge(c,d));
90 }

```

4 Flow

4.1 dinic

```

1 template<class T>
2 struct Dinic{
3     struct edge{
4         int from, to;
5         T cap;
6         edge(int _from, int _to, T _cap) : from(
            _from), to(_to), cap(_cap) {}
7     };
8     int n;
9     vector<edge> edges;
10    vector<vector<int>> g;
11    vector<int> cur, h;
12    Dinic(int _n) : n(_n+1), g(_n+1) {}
13    void add_edge(int u, int v, T cap){

```

```

14    g[u].push_back(edges.size());
15    edges.push_back(edge(u, v, cap));
16    g[v].push_back(edges.size());
17    edges.push_back(edge(v, u, 0));
18 }
19 bool bfs(int s,int t){
20     h.assign(n, -1);
21     h[s] = 0;
22     queue<int> que;
23     que.push(s);
24     while(!que.empty()) {
25         int u = que.front();
26         que.pop();
27         for(auto id : g[u]) {
28             const edge& e = edges[id];
29             int v = e.to;
30             if(e.cap > 0 && h[v] == -1) {
31                 h[v] = h[u] + 1;
32                 if(v == t) {
33                     return 1;
34                 }
35                 que.push(v);
36             }
37         }
38     }
39     return 0;
40 }
41 T dfs(int u, int t, T f) {
42     if(u == t) {
43         return f;
44     }
45     T r = f;
46     for(int& i = cur[u]; i < (int) g[u].size
        (); ++i) {
47         int id = g[u][i];
48         const edge& e = edges[id];
49         int v = e.to;
50         if(e.cap > 0 && h[v] == h[u] + 1) {
51             T send = dfs(v, t, min(r, e.cap));
52             edges[id].cap -= send;
53             edges[id ^ 1].cap += send;
54             r -= send;
55             if(r == 0) {
56                 return f;
57             }
58         }
59     }
60     return f - r;
61 }
62 T flow(int s, int t, T f = numeric_limits<
    T>::max()) {
63     T ans = 0;
64     while(f > 0 && bfs(s, t)) {
65         cur.assign(n, 0);
66         T send = dfs(s, t, f);
67         ans += send;
68         f -= send;
69     }
70     return ans;
71 }
72 vector<pair<int,int>> min_cut(int s) {
73     vector<bool> vis(n);
74     vis[s] = true;
75     queue<int> que;
76     que.push(s);
77     while(!que.empty()) {

```

```

78     int u = que.front();
79     que.pop();
80     for(auto id : g[u]) {
81         const auto& e = edges[id];
82         int v = e.to;
83         if(e.cap > 0 && !vis[v]) {
84             vis[v] = true;
85             que.push(v);
86         }
87     }
88 }
89 vector<pair<int,int>> cut;
90 for(int i = 0; i < (int) edges.size(); i
    += 2) {
91     const auto& e = edges[i];
92     if(vis[e.from] && !vis[e.to]) {
93         cut.push_back(make_pair(e.from, e.to
            ));
94     }
95 }
96 return cut;
97 }
98 };

```

4.2 Gomory Hu

```

1 //最小割樹+求任兩點間最小割
2 //0-base, root=0
3 LL e[MXN][MXN]; //任兩點間最小割
4 int p[MXN]; //parent
5 ISAP D; // original graph
6 void gomory_hu(){
7     fill(p, p+n, 0);
8     fill(e[0], e[n], INF);
9     for( int s = 1; s < n; ++s ) {
10         int t = p[s];
11         ISAP F = D;
12         LL tmp = F.min_cut(s, t);
13         for( int i = 1; i < s; ++i )
14             e[s][i] = e[i][s] = min(tmp, e[t][i]);
15         for( int i = s+1; i <= n; ++i )
16             if( p[i] == t && F.vis[i] ) p[i] = s;
17     }
18 }

```

4.3 ISAP with cut

```

1 template<typename T>
2 struct ISAP{
3     static const int MAXN=105;
4     static const T INF=INT_MAX;
5     int n;//點數
6     int d[MAXN],gap[MAXN],cur[MAXN];
7     struct edge{
8         int v,pre;
9         T cap,r;
10         edge(int v,int pre,T cap):v(v),pre(pre),
            cap(cap),r(cap){}
11 };

```

```

12 int g[MAXN];
13 vector<edge> e;
14 void init(int _n){
15     memset(g,-1,sizeof(int)*((n=_n)+1));
16     e.clear();
17 }
18 void add_edge(int u,int v,T cap,bool
    directed=false){
19     e.push_back(edge(v,g[u],cap));
20     g[u]=e.size()-1;
21     e.push_back(edge(u,g[v],directed?0:cap));
22     ;
23     g[v]=e.size()-1;
24 }
25 T dfs(int u,int s,int t,T CF=INF){
26     if(u==t)return CF;
27     T tf=CF,df;
28     for(int &i=cur[u];~i;i=e[i].pre){
29         if(e[i].r&&d[u]==d[e[i].v]+1){
30             df=dfs(e[i].v,s,t,min(tf,e[i].r));
31             e[i].r-=df;
32             e[i^1].r+=df;
33             if(!(tf==df)||d[s]==n)return CF-tf;
34         }
35     }
36     int mh=n;
37     for(int i=cur[u]=g[u];~i;i=e[i].pre){
38         if(e[i].r&&d[e[i].v]<mh)mh=d[e[i].v];
39     }
40     if(!--gap[d[u]]d[s]=n;
41     else ++gap[d[u]]+=mh;
42     return CF-tf;
43 }
44 T isap(int s,int t,bool clean=true){
45     memset(d,0,sizeof(int)*(n+1));
46     memset(gap,0,sizeof(int)*(n+1));
47     memcpy(cur,g,sizeof(int)*(n+1));
48     if(clean) for(size_t i=0;i<e.size();++i)
49         e[i].r=e[i].cap;
50     T MF=0;
51     for(gap[0]=n;d[s]<n;MF+=dfs(s,s,t);
52     return MF;
53 }
54 vector<int> cut_e;//最小割邊集
55 bool vis[MXN];
56 void dfs_cut(int u){
57     vis[u]=1;//表示u屬於source的最小割集
58     for(int i=g[u];~i;i=e[i].pre)
59         if(e[i].r>0&&!vis[e[i].v])dfs_cut(e[i
            ].v);
60 }
61 T min_cut(int s,int t){
62     T ans=isap(s,t);
63     memset(vis,0,sizeof(bool)*(n+1));
64     dfs_cut(s), cut_e.clear();
65     for(int u=0;u<n;++u)if(vis[u])
66         for(int i=g[u];~i;i=e[i].pre)
67             if(!vis[e[i].v])cut_e.push_back(i);
68     return ans;
69 };

```

4.4 MinCostMaxFlow

```

1 template<class Cap_t, class Cost_t>
2 class MCMF {
3 public:
4     struct Edge {
5         int from;
6         int to;
7         Cap_t cap;
8         Cost_t cost;
9         Edge(int u, int v, Cap_t _cap, Cost_t
            _cost) : from(u), to(v), cap(_cap),
            cost(_cost) {}
10    };
11
12    static constexpr Cap_t EPS = static_cast<
        Cap_t>(1e-9);
13
14    int n;
15    vector<Edge> edges;
16    vector<vector<int>> g;
17    vector<Cost_t> d;
18    vector<bool> in_queue;
19    vector<int> previous_edge;
20
21    MCMF() {}
22    MCMF(int _n) : n(_n+1), g(_n+1), d(_n+1),
        in_queue(_n+1), previous_edge(_n+1) {}
23
24    void add_edge(int u, int v, Cap_t cap,
        Cost_t cost) {
25        assert(0 <= u && u < n);
26        assert(0 <= v && v < n);
27        g[u].push_back(edges.size());
28        edges.emplace_back(u, v, cap, cost);
29        g[v].push_back(edges.size());
30        edges.emplace_back(v, u, 0, -cost);
31    }
32
33    bool spfa(int s, int t) {
34        bool found = false;
35        fill(d.begin(), d.end(), numeric_limits<
            Cost_t>::max());
36        d[s] = 0;
37        in_queue[s] = true;
38        queue<int> que;
39        que.push(s);
40        while(!que.empty()) {
41            int u = que.front();
42            que.pop();
43            if(u == t) {
44                found = true;
45            }
46            in_queue[u] = false;
47            for(auto& id : g[u]) {
48                const Edge& e = edges[id];
49                if(e.cap > EPS && d[u] + e.cost < d[
                    e.to]) {
50                    d[e.to] = d[u] + e.cost;
51                    previous_edge[e.to] = id;
52                    if(!in_queue[e.to]) {
53                        que.push(e.to);
54                        in_queue[e.to] = true;
55                    }
56                }

```

```

57    }
58    }
59    return found;
60 }
61
62 pair<Cap_t, Cost_t> flow(int s, int t,
    Cap_t f = numeric_limits<Cap_t>::max())
63 {
64     assert(0 <= s && s < n);
65     assert(0 <= t && t < n);
66     Cap_t cap = 0;
67     Cost_t cost = 0;
68     while(f > 0 && spfa(s, t)) {
69         Cap_t send = f;
70         int u = t;
71         while(u != s) {
72             const Edge& e = edges[previous_edge[
                u]];
73             send = min(send, e.cap);
74             u = e.from;
75         }
76         u = t;
77         while(u != s) {
78             Edge& e = edges[previous_edge[u]];
79             e.cap -= send;
80             Edge& b = edges[previous_edge[u] ^
                1];
81             b.cap += send;
82             u = e.from;
83         }
84         cap += send;
85         f -= send;
86         cost += send * d[t];
87     }
88     return make_pair(cap, cost);
89 };

```

4.5 Property

- 1 最大流 = 最小割
- 2 最大獨立集 = 補圖最大團 = V - 最小頂點覆蓋
- 3 二分圖最大匹配 = 二分圖最小頂點覆蓋
- 4 二分圖最大匹配加s,t點 = 最大流

5 Graph

5.1 2-SAT

```

1 struct two_sat{
2     SCC s;
3     vector<bool>ans;
4     int have_ans = 0;
5     int n;
6     two_sat(int _n) : n(_n) {
7         ans.resize(n+1);
8         s = SCC(2*n);

```

```

9 }
10 int inv(int x){
11     if(x>n)return x-n;
12     return x+n;
13 }
14 void add_or_clause(int u, bool x, int v,
15     bool y){
16     if(!x)u = inv(u);
17     if(!y)v = inv(v);
18     s.add_edge(inv(u), v);
19     s.add_edge(inv(v), u);
20 }
21 void check(){
22     if(have_ans!=0)return;
23     s.build();
24     for(int i = 0; i<n; ++i){
25         if(s.scc[i]==s.scc[inv(i)]){
26             have_ans = -1;
27             return;
28         }
29         ans[i] = (s.scc[i]<s.scc[inv(i)]);
30     }
31     have_ans = 1;
32 };

```

5.2 count-bridge-online

```

1 vector<int> par, dsu_2ecc, dsu_cc,
2     dsu_cc_size, last_visit;
3 int bridges, lca_iteration;
4 void init(int n) {
5     par.assign(n, -1);
6     dsu_2ecc.resize(n);
7     dsu_cc.resize(n);
8     dsu_cc_size.assign(n, 1);
9     lca_iteration = 0;
10    last_visit.assign(n, 0);
11    iota(ALL(dsu_cc), 0);
12    dsu_2ecc = dsu_cc;
13    bridges = 0;
14 }
15 int find_2ecc(int v) {
16     if(v == -1) return -1;
17     return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v]);
18 }
19 int find_cc(int v) {
20     v = find_2ecc(v);
21     return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
22 }
23 void make_root(int v) {
24     v = find_2ecc(v);
25     int root = v, child = -1;
26     while(v != -1) {
27         int p = find_2ecc(par[v]);
28         par[v] = child;
29         dsu_cc[v] = root;
30         child = v;
31         v = p;
32     }
33     dsu_cc_size[root] = dsu_cc_size[child];
34 }
35 void merge_path(int a, int b) {
36     ++lca_iteration;
37     vector<int> path_a, path_b;
38     int lca = -1;
39     while(lca == -1) {
40         if(a != -1) {
41             a = find_2ecc(a);
42             path_a.push_back(a);
43             if(last_visit[a] == lca_iteration){
44                 lca = a;
45                 break;
46             }
47             last_visit[a] = lca_iteration;
48             a = par[a];
49         }
50         if(b != -1) {
51             b = find_2ecc(b);
52             path_b.push_back(b);
53             if(last_visit[b] == lca_iteration){
54                 lca = b;
55                 break;
56             }
57             last_visit[b] = lca_iteration;
58             b = par[b];
59         }
60     }
61     for(int v : path_a) {
62         dsu_2ecc[v] = lca;
63         if(v == lca) break;
64         --bridges;
65     }
66     for(int v : path_b) {
67         dsu_2ecc[v] = lca;
68         if(v == lca) break;
69         --bridges;
70     }
71 }
72 void add_edge(int a, int b) {
73     a = find_2ecc(a), b = find_2ecc(b);
74     if(a == b) return;
75     int ca = find_cc(a), cb = find_cc(b);
76     if(ca != cb) {
77         ++bridges;
78         if(dsu_cc_size[ca] > dsu_cc_size[cb]) swap(a, b), swap(ca, cb);
79         make_root(a);
80         par[a] = dsu_cc[a] = b;
81         dsu_cc_size[cb] += dsu_cc_size[a];
82     } else merge_path(a, b);
83 }

```

5.3 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;

```

5.4 manhattan-mst

```

1 void solve(Point *a, int n) {
2     sort(a, a + n, [](const Point &p, const
3         Point &q) {
4             return p.x + p.y < q.x + q.y;
5         });
6     set<Point> st; // greater<Point::x>
7     for (int i = 0; i < n; ++i) {
8         for (auto it = st.lower_bound(a[i]);
9             it != st.end(); it = st.erase(

```

```

10             it)) {
11                 if (it -> x - it -> y < a[i].x -
12                     a[i].y) break;
13                 es.push_back({it -> u, a[i].u,
14                     dist(*it, a[i])});
15             }
16             st.insert(a[i]);
17         }
18     }
19 void MST(Point *a, int n) {
20     for (int t = 0; t < 2; ++t) {
21         solve(a, n);
22         for (int i = 0; i < n; ++i) swap(a[i]
23             .x, a[i].y);
24         solve(a, n);
25         for (int i = 0; i < n; ++i) a[i].x =
26             -a[i].x;
27     }
28 }

```

5.5 Minimum Clique Cover

```

1 struct Clique_Cover { // 0-base, O(n2^n)
2     int co[1 << N], n, E[N];
3     int dp[1 << N];
4     void init(int _n) {
5         n = _n, fill_n(dp, 1 << n, 0);
6         fill_n(E, n, 0), fill_n(co, 1 << n, 0);
7     }
8     void add_edge(int u, int v) {
9         E[u] |= 1 << v, E[v] |= 1 << u;
10    }
11    int solve() {
12        for (int i = 0; i < n; ++i)
13            co[1 << i] = E[i] | (1 << i);
14        co[0] = (1 << n) - 1;
15        dp[0] = (n & 1) * 2 - 1;
16        for (int i = 1; i < (1 << n); ++i) {
17            int t = i & -i;
18            dp[i] = -dp[i ^ t];
19            co[i] = co[i ^ t] & co[t];
20        }
21        for (int i = 0; i < (1 << n); ++i)
22            co[i] = (co[i] & i) == i;
23        fwt(co, 1 << n, 1);
24        for (int ans = 1; ans < n; ++ans) {
25            int sum = 0; // probabilistic
26            for (int i = 0; i < (1 << n); ++i)
27                sum += (dp[i] * co[i]);
28            if (sum) return ans;
29        }
30        return n;
31    }
32 };

```

5.6 SCC

```

1 struct SCC{
2     int n, cnt = 0, dfn_cnt = 0;
3     vector<vector<int>>g;

```

```

4 vector<int>sz, scc, low, dfn;
5 stack<int>st;
6 vector<bool>vis;
7 SCC(int _n = 0) : n(_n){
8     sz.resize(n+5), scc.resize(n+5), low.
        resize(n+5), dfn.resize(n+5), vis.
        resize(n+5);
9     g.resize(n+5);
10 }
11 inline void add_edge(int u, int v){
12     g[u].push_back(v);
13 }
14 inline void build(){
15     function<void(int, int)>dfs = [&](int u,
        int dis){
16         low[u] = dfn[u] = ++dfn_cnt, vis[u] =
            1;
17         st.push(u);
18         for(auto v: g[u]){
19             if(!dfn[v]){
20                 dfs(v, dis+1);
21                 low[u] = min(low[u], low[v]);
22             }
23             else if(vis[v]){
24                 low[u] = min(low[u], dfn[v]);
25             }
26         }
27         if(low[u]==dfn[u]){
28             ++cnt;
29             while(vis[u]){
30                 auto v = st.top();
31                 st.pop();
32                 vis[v] = 0;
33                 scc[v] = cnt;
34                 sz[cnt]++;
35             }
36         }
37     };
38     for(int i = 0; i<n; ++i){
39         if(!scc[i]){
40             dfs(i, 1);
41         }
42     }
43 }
44 vector<vector<int>> compress(){
45     vector<vector<int>>ans(cnt+1);
46     for(int u = 0; u<n; ++u){
47         for(auto v: g[u]){
48             if(scc[u] == scc[v]){
49                 continue;
50             }
51             ans[scc[u]].push_back(scc[v]);
52         }
53     }
54     for(int i = 0; i<cnt; ++i){
55         sort(ans[i].begin(), ans[i].end());
56         ans[i].erase(unique(ans[i].begin(),
            ans[i].end()), ans[i].end());
57     }
58     return ans;
59 }
60 };

```

5.7 判斷環

```

1 vector<int> G[MAXN];
2 bool visit[MAXN];
3 /* return if the connected component where u
   is
   contains a cycle*/
4 bool dfs(int u, int pre) {
5     if(visit[u]) return true;
6     visit[u] = true;
7     for(int v : G[u])
8         if(v != pre && dfs(v, u))
9             return true;
10    return false;
11 }
12 //check if a graph contains a cycle
13 bool checkCycle(int n) {
14     for(int i = 1; i <= n; i++)
15         if(!visit[i] && dfs(i, -1))
16             return true;
17     return false;
18 }

```

5.8 最大團

```

1 struct MaxClique{
2     static const int MAXN=105;
3     int N, ans;
4     int g[MAXN][MAXN], dp[MAXN], stk[MAXN][MAXN];
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            }
20            else return 0;
21        }
22        for(int i=0; i<ns; ++i){
23            if(dep+ns-i<ans) return 0;
24            int u=stk[dep][i], cnt=0;
25            if(dep+dp[u]<=ans) return 0;
26            for(int j=i+1; j<ns; ++j){
27                int v=stk[dep][j];
28                if(g[u][v]) stk[dep+1][cnt++]=v;
29            }
30            tmp[dep]=u;
31            if(dfs(cnt, dep+1)) return 1;
32        }
33        return 0;
34    }

```

```

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 枚舉極大團 Bron-Kerbosch

```

1 //O(3^n / 3)
2 struct maximalCliques{
3     using Set = vector<int>;
4     size_t n; //1-base
5     vector<Set> G;
6     static Set setUnion(const Set &A, const
        Set &B){
7         Set C(A.size() + B.size());
8         auto it = set_union(A.begin(), A.end(), B.
            begin(), B.end(), C.begin());
9         C.erase(it, C.end());
10        return C;
11    }
12    static Set setIntersection(const Set &A,
        const Set &B){
13        Set C(min(A.size(), B.size()));
14        auto it = set_intersection(A.begin(), A.
            end(), B.begin(), B.end(), C.begin());
15        C.erase(it, C.end());
16        return C;
17    }
18    static Set setDifference(const Set &A,
        const Set &B){
19        Set C(min(A.size(), B.size()));
20        auto it = set_difference(A.begin(), A.
            end(), B.begin(), B.end(), C.begin());
21        C.erase(it, C.end());
22        return C;
23    }
24    void BronKerbosch1(Set R, Set P, Set X){
25        if(P.empty() && X.empty()){
26            // R form an maximal clique
27            return;
28        }
29        for(auto v: P){
30            BronKerbosch1(setUnion(R, {v}),
                setIntersection(P, G[v]),
                setIntersection(X, G[v]));
31            P = setDifference(P, {v});
32            X = setUnion(X, {v});
33        }
34    }
35    void init(int _n){
36        G.clear();
37        G.resize((n = _n) + 1);
38    }
39    void addEdge(int u, int v){
40        G[u].emplace_back(v);
41        G[v].emplace_back(u);
42    }

```

```

42 }
43 void solve(int n){
44     Set P;
45     for(int i=1; i<=n; ++i){
46         sort(G[i].begin(), G[i].end());
47         G[i].erase(unique(G[i].begin(), G[i].end()),
            G[i].end());
48         P.emplace_back(i);
49     }
50     BronKerbosch1({}, P, {});
51 }
52 };
53 //判斷圖G是否能3染色：
54 //枚舉圖G的極大獨立集I (極大獨立集 = 補圖極
    大團)
55 //若存在I使得G-I形成二分圖，則G可以三染色
56 //反之則不能3染色
57

```

5.10 橋連通分量

```

1 vector<pii> findBridges(const vector<vector<
    int>>& g) {
2     int n = (int) g.size();
3     vector<int> id(n, -1), low(n);
4     vector<pii> bridges;
5     function<void(int, int)> dfs = [&](int u,
        int p) {
6         static int cnt = 0;
7         id[u] = low[u] = cnt++;
8         for(auto v : g[u]) {
9             if(v == p) continue;
10            if(id[v] != -1) low[u] = min(low[u],
                id[v]);
11            else {
12                dfs(v, u);
13                low[u] = min(low[u], low[v]);
14                if(low[v] > id[u]) bridges.emplace_back(u, v);
15            }
16        }
17    };
18    for(int i = 0; i < n; ++i) {
19        if(id[i] == -1) dfs(i, -1);
20    }
21    return bridges;
22 }

```

5.11 雙連通分量 & 割點

```

1 struct BCC_AP{
2     int dfn_cnt = 0, bcc_cnt = 0, n;
3     vector<int> dfn, low, ap, bcc_id;
4     stack<int> st;
5     vector<bool> vis, is_ap;
6     vector<vector<int>> bcc;
7     BCC_AP(int _n): n(_n){
8         dfn.resize(n+5), low.resize(n+5), bcc.
            resize(n+5), vis.resize(n+5), is_ap.
            resize(n+5), bcc_id.resize(n+5);
9     }

```



```

9 }
10 inline void build(const vector<vector<int>
    >>&g,int u,int p = -1){
11     int child = 0;
12     dfn[u] = low[u] = ++dfn_cnt;
13     st.push(u);
14     vis[u] = 1;
15     if(g[u].empty() and p==-1){
16         bcc_id[u] = ++bcc_cnt;
17         bcc[bcc_cnt].push_back(u);
18         return;
19     }
20     for(auto v:g[u]){
21         if(v==p)continue;
22         if(!dfn[v]){
23             build(g,v,u);
24             child++;
25             if(dfn[u]<=low[v]){
26                 is_ap[u] = 1;
27                 bcc_id[u] = ++bcc_cnt;
28                 bcc[bcc_cnt].push_back(u);
29                 while(vis[v]){
30                     bcc_id[st.top()] = bcc_cnt;
31                     bcc[bcc_cnt].push_back(st.top())
32                     ;
33                     vis[st.top()] = 0;
34                     st.pop();
35                 }
36                 low[u] = min(low[u],low[v]);
37             }
38             low[u] = min(low[u],dfn[v]);
39         }
40         if(p==-1 and child<2)is_ap[u] = 0;
41         if(is_ap[u])ap.push_back(u);
42     }
43 };

```

6 Math

6.1 Basic

```

1 template<typename T>
2 void gcd(const T &a,const T &b,T &d,T &x,T &y){
3     if(!b) d=a,x=1,y=0;
4     else gcd(b,a%b,d,y,x), y=-x*(a/b);
5 }
6 long long int phi[N+1];
7 void phiTable(){
8     for(int i=1;i<=N;i++)phi[i]=i;
9     for(int i=1;i<=N;i++)for(x=i*2;x<=N;x+=i)
10         phi[x]-=phi[i];
11 }
12 void all_divdown(const LL &n) { // all n/x
13     for(LL a=1;a<=n;a=n/(n/(a+1))) {
14         // dosomething;
15     }
16 }
17 const int MAXPRIME = 1000000;

```

```

17 int iscom[MAXPRIME], prime[MAXPRIME],
    primecnt;
18 int phi[MAXPRIME], mu[MAXPRIME];
19 void sieve(void){
20     memset(iscom,0,sizeof(iscom));
21     primecnt = 0;
22     phi[1] = mu[1] = 1;
23     for(int i=2;i<MAXPRIME;++i) {
24         if(!iscom[i]) {
25             prime[primecnt++] = i;
26             mu[i] = -1;
27             phi[i] = i-1;
28         }
29         for(int j=0;j<primecnt;++j) {
30             int k = i * prime[j];
31             if(k>MAXPRIME) break;
32             iscom[k] = prime[j];
33             if(i%prime[j]==0) {
34                 mu[k] = 0;
35                 phi[k] = phi[i] * prime[j];
36                 break;
37             } else {
38                 mu[k] = -mu[i];
39                 phi[k] = phi[i] * (prime[j]-1);
40             }
41         }
42     }
43 }
44 }
45 bool g_test(const LL &g, const LL &p, const
    vector<LL> &v) {
46     for(int i=0;i<v.size();++i)
47         if(modexp(g,(p-1)/v[i],p)==1)
48             return false;
49     return true;
50 }
51 LL primitive_root(const LL &p) {
52     if(p==2) return 1;
53     vector<LL> v;
54     Factor(p-1,v);
55     v.erase(unique(v.begin(), v.end()), v.end
        ());
56     for(LL g=2;g<p;++g)
57         if(g_test(g,p,v))
58             return g;
59     puts("primitive_root NOT FOUND");
60     return -1;
61 }
62 int Legendre(const LL &a, const LL &p) {
63     return modexp(a%p,(p-1)/2,p);
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 int inv[maxN];
71 LL invtable(int n,LL P){
72     inv[1]=1;
73     for(int i=2;i<n;++i)
74         inv[i]=(P-(P/i))*inv[P%i]%P;
75 }
76 LL log_mod(const LL &a, const LL &b, const
    LL &p) {

```

```

78 // a ^ x = b ( mod p )
79 int m=sqrt(p+.5), e=1;
80 LL v=inv(modexp(a,m,p), p);
81 map<LL,int> x;
82 x[1]=0;
83 for(int i=1;i<m;++i) {
84     e = LLMul(e,a,p);
85     if(!x.count(e)) x[e] = i;
86 }
87 for(int i=0;i<m;++i) {
88     if(x.count(b)) return i*m + x[b];
89     b = LLMul(b,v,p);
90 }
91 return -1;
92 }
93 }
94 LL Tonelli_Shanks(const LL &n, const LL &p)
    {
95     // x^2 = n ( mod p )
96     if(n==0) return 0;
97     if(Legendre(n,p)!=1) while(1) { puts("SQRT
        ROOT does not exist"); }
98     int S = 0;
99     LL Q = p-1;
100     while( !(Q&1) ) { Q>>=1; ++S; }
101     if(S==1) return modexp(n%p,(p+1)/4,p);
102     LL z = 2;
103     for(;Legendre(z,p)!=-1;++z)
104         LL c = modexp(z,Q,p);
105         LL R = modexp(n%p,(Q+1)/2,p), t = modexp(n
            %p,Q,p);
106         int M = S;
107         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*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;k>>=1){
136         if(k&1)ans=ans*n%m;
137         n=n*n%m;
138     }
139     return ans;
140 }

```

```

141 template<typename T>
142 T crt(vector<T> &m,vector<T> &a){
143     T M=1,tM=ans=0;
144     for(int i=0;i<(int)m.size();++i)M*=m[i];
145     for(int i=0;i<(int)a.size();++i){
146         tM=M/m[i];
147         ans=(ans+(a[i]*tM%M)*pow_mod(tM,Euler(m[
            i])-1,m[i])%M)%M;
148         /*如果m[i]是質數 · Euler(m[i])-1=m[i]-2 ·
            就不用算Euler了*/
149     }
150     return ans;
151 }

```

6.2 Bit Set

```

1 void sub_set(int S){
2     int sub=S;
3     do{
4         //對某集合的子集的處理
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的子集的處理
12        int x=comb&-comb,y=comb+x;
13        comb=((comb&y)/x>>1)|y;
14    }
15 }

```

6.3 Combination

```

1 mint binom(int n, int k) {
2     if(k < 0 || k > n) return 0;
3     return fact[n] * inv_fact[k] * inv_fact[n
        - k];
4 }
5 // a_1 + a_2 + ... + a_n = k, a_i >= 0
6 mint stars_and_bars(int n, int k) { return
    binom(k + n - 1, n - 1); }
7 // number of ways from (0, 0) to (n, m)
8 mint paths(int n, int m) { return binom(n +
    m, n); }
9 mint catalan(int n) { return binom(2 * n, n)
    - binom(2 * n, n + 1); }

```

6.4 ExtendGCD

```

1 // ax + by = gcd(a, b)
2 ll ext_gcd(ll a, ll b, ll& x, ll& y) {
3     if(b == 0) {
4         x = 1, y = 0;
5         return a;
6     }

```

```

7 | ll x1, y1;
8 | ll g = ext_gcd(b, a % b, x1, y1);
9 | x = y1, y = x1 - (a / b) * y1;
10 | return g;
11 | }

```

6.5 FFT

```

1 | // Fast-Fourier-Transform
2 | using cd = complex<double>;
3 | const double PI = acos(-1);
4 |
5 | void FFT(vector<cd>& a, bool inv) {
6 |     int n = (int) a.size();
7 |     for(int i = 1, j = 0; i < n; ++i) {
8 |         int bit = n >> 1;
9 |         for(; j & bit; bit >>= 1) {
10 |             j ^= bit;
11 |         }
12 |         j ^= bit;
13 |         if(i < j) {
14 |             swap(a[i], a[j]);
15 |         }
16 |     }
17 |     for(int len = 2; len <= n; len <= 1) {
18 |         const double ang = 2 * PI / len * (inv ?
19 |             -1 : +1);
20 |         cd rot(cos(ang), sin(ang));
21 |         for(int i = 0; i < n; i += len) {
22 |             cd w(1);
23 |             for(int j = 0; j < len / 2; ++j) {
24 |                 cd u = a[i + j], v = a[i + j + len /
25 |                     2] * w;
26 |                 a[i + j] = u + v;
27 |                 a[i + j + len / 2] = u - v;
28 |                 w *= rot;
29 |             }
30 |         }
31 |         if(inv) {
32 |             for(auto& x : a) {
33 |                 x /= n;
34 |             }
35 |         }
36 |     }
37 |     vector<int> multiply(const vector<int>& a,
38 |         const vector<int>& b) {
39 |         vector<cd> fa(a.begin(), a.end());
40 |         vector<cd> fb(b.begin(), b.end());
41 |         int n = 1;
42 |         while(n < (int) a.size() + (int) b.size()
43 |             - 1) {
44 |             n <= 1;
45 |         }
46 |         fa.resize(n);
47 |         fb.resize(n);
48 |         FFT(fa, false);
49 |         FFT(fb, false);
50 |         for(int i = 0; i < n; ++i) {
51 |             fa[i] *= fb[i];
52 |         }
53 |         FFT(fa, true);

```

```

52 | vector<int> c(a.size() + b.size() - 1);
53 | for(int i = 0; i < (int) c.size(); ++i) {
54 |     c[i] = round(fa[i].real());
55 | }
56 | return c;
57 | }

```

6.6 FWT

```

1 | vector<int> F_OR_T(vector<int> f, bool
2 |     inverse){
3 |     for(int i=0; (2<<i)<=f.size(); ++i)
4 |         for(int j=0; j<f.size(); j+=2<<i)
5 |             for(int k=0; k<(1<<i); ++k)
6 |                 f[j+k+(1<<i)] += f[j+k]*(inverse
7 |                     ?-1:1);
8 |     return f;
9 | }
10 | vector<int> rev(vector<int> A) {
11 |     for(int i=0; i<A.size(); i+=2)
12 |         swap(A[i], A[i^(A.size()-1)]);
13 |     return A;
14 | }
15 | vector<int> F_AND_T(vector<int> f, bool
16 |     inverse){
17 |     return rev(F_OR_T(rev(f), inverse));
18 | }
19 | vector<int> F_XOR_T(vector<int> f, bool
20 |     inverse){
21 |     for(int i=0; (2<<i)<=f.size(); ++i)
22 |         for(int j=0; j<f.size(); j+=2<<i)
23 |             for(int k=0; k<(1<<i); ++k){
24 |                 int u=f[j+k], v=f[j+k+(1<<i)];
25 |                 f[j+k+(1<<i)] = u-v, f[j+k] = u+v;
26 |             }
27 |     if(inverse) for(auto &a:f) a/=f.size();
28 |     return f;
29 | }

```

6.7 Gauss-Jordan

```

1 | int GaussJordan(vector<vector<ld>>& a) {
2 |     // -1 no sol, 0 inf sol
3 |     int n = SZ(a);
4 |     REP(i, n) assert(SZ(a[i]) == n + 1);
5 |     REP(i, n) {
6 |         int p = i;
7 |         REP(j, n) {
8 |             if(j < i && abs(a[j][i]) > EPS)
9 |                 continue;
10 |             if(abs(a[j][i]) > abs(a[p][i])) p = j;
11 |         }
12 |         REP(j, n + 1) swap(a[i][j], a[p][j]);
13 |         if(abs(a[i][i]) <= EPS) continue;
14 |         REP(j, n) {
15 |             if(i == j) continue;
16 |             ld delta = a[j][i] / a[i][i];
17 |             FOR(k, i, n + 1) a[j][k] -= delta * a[i][k];

```

```

18 |     }
19 |     bool ok = true;
20 |     REP(i, n) {
21 |         if(abs(a[i][i]) <= EPS) {
22 |             if(abs(a[i][n]) > EPS) return -1;
23 |             ok = false;
24 |         }
25 |     }
26 |     return ok;
27 | }

```

6.8 GCD-Convolution

```

1 | // 2, 3, 5, 7, ...
2 | vector<int> prime_enumerate(int N) {
3 |     vector<bool> sieve(N / 3 + 1, 1);
4 |     for(int p = 5, d = 4, i = 1, sqn = sqrt(N);
5 |         p <= sqn; p += d = 6 - d, i++) {
6 |         if(!sieve[i]) continue;
7 |         for(int q = p * p / 3, r = d * p / 3 + (
8 |             d * p % 3 == 2), s = 2 * p; q < SZ(
9 |                 sieve); q += r = s - r) sieve[q] =
10 |                     0;
11 |     }
12 |     vector<int> ret{2, 3};
13 |     for(int p = 5, d = 4, i = 1; p <= N; p +=
14 |         d = 6 - d, i++) {
15 |         if(sieve[i]) {
16 |             ret.pb(p);
17 |         }
18 |     }
19 |     while(SZ(ret) && ret.back() > N) ret.
20 |         pop_back();
21 |     return ret;
22 | }
23 | struct divisor_transform {
24 |     template<class T>
25 |     static void zeta_transform(vector<T>& a) {
26 |         int n = a.size() - 1;
27 |         for(auto p : prime_enumerate(n)) {
28 |             for(int i = 1; i * p <= n; i++) {
29 |                 a[i * p] += a[i];
30 |             }
31 |         }
32 |     }
33 |     template<class T>
34 |     static void mobius_transform(vector<T>& a) {
35 |         int n = a.size() - 1;
36 |         for(auto p : prime_enumerate(n)) {
37 |             for(int i = n / p; i > 0; i--) {
38 |                 a[i * p] -= a[i];
39 |             }
40 |         }
41 |     }
42 | };
43 | struct multiple_transform {
44 |     template<class T>
45 |     static void zeta_transform(vector<T>& a) {
46 |         int n = a.size() - 1;
47 |         for(auto p : prime_enumerate(n)) {
48 |             for(int i = n / p; i > 0; i--) {
49 |                 a[i] += a[i * p];

```

```

44 |     }
45 | }
46 | template<class T>
47 | static void mobius_transform(vector<T>& a) {
48 |     {
49 |         int n = a.size() - 1;
50 |         for(auto p : prime_enumerate(n)) {
51 |             for(int i = 1; i * p <= n; i++) {
52 |                 a[i] -= a[i * p];
53 |             }
54 |         }
55 |     }
56 | };
57 | // lcm: multiple -> divisor
58 | template<class T>
59 | vector<T> gcd_convolution(const vector<T>& a
60 |     , const vector<T>& b) {
61 |     assert(a.size() == b.size());
62 |     auto f = a, g = b;
63 |     multiple_transform::zeta_transform(f);
64 |     multiple_transform::zeta_transform(g);
65 |     REP(i, SZ(f)) f[i] *= g[i];
66 |     multiple_transform::mobius_transform(f);
67 |     return f;

```

6.9 InvGCD

```

1 | pair<long long, long long> inv_gcd(long long
2 |     a, long long b) {
3 |     a %= b;
4 |     if(a < 0) a += b;
5 |     if(a == 0) return {b, 0};
6 |     long long s = b, t = a;
7 |     long long m0 = 0, m1 = 1;
8 |     while(t) {
9 |         long long u = s / t;
10 |         s -= t * u;
11 |         m0 -= m1 * u;
12 |         swap(s, t);
13 |         swap(m0, m1);
14 |     }
15 |     if(m0 < 0) m0 += b / s;
16 |     return {s, m0};

```

6.10 LinearCongruence

```

1 | pair<LL, LL> LinearCongruence(LL a[], LL b[],
2 |     LL m[], int n) {
3 |     // a[i]*x = b[i] (mod m[i])
4 |     for(int i=0; i<n; ++i) {
5 |         LL x, y, d = extgcd(a[i], m[i], x, y);
6 |         if(b[i]%d!=0) return make_pair(-1LL, 0LL);
7 |         m[i] /= d;
8 |         b[i] = LLmul(b[i]/d, x, m[i]);
9 |     }
10 |     LL lastb = b[0], lastm = m[0];

```

```

10 for(int i=1;i<n;++i) {
11     LL x, y, d = extgcd(m[i],lastm,x,y);
12     if((lastb-b[i])%d!=0) return make_pair
13         (-1LL,0LL);
14     lastb = Llmul((lastb-b[i])/d,x,(lastm/d)
15         )*m[i];
16     lastm = (lastm/d)*m[i];
17     lastb = (lastb+b[i])%lastm;
18 }

```

6.11 Lucas

```

1 ll C(ll n, ll m, ll p){// n!/m!/(n-m)!
2 if(n<m) return 0;
3 return f[n]*inv(f[m],p)%p*inv(f[n-m],p)%p;
4 }
5 ll L(ll n, ll m, ll p){
6 if(!m) return 1;
7 return C(n%p,m%p,p)*L(n/p,m/p,p)%p;
8 }
9 ll Wilson(ll n, ll p){ // n!%p
10 if(!n)return 1;
11 ll res=Wilson(n/p, p);
12 if((n/p)%2) return res*(p-f[n%p])%p;
13 return res*f[n%p]%p; //(p-1)!%p=-1
14 }

```

6.12 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            for(auto &j:m[i])j/=lazy[i];
74        }
75        return det;
76    }
77 };

```

6.13 NTT

```

1 const ll mod = (119 << 23) + 1, root = 62;
2 // = 998244353
3 // For p < 2^30 there is also e.g. 5 << 25,
4 // 7 << 26, 479 << 21
5 // and 483 << 21 (same root). The last two
6 // are > 10^9.
7 typedef vector<ll> vl;
8 void ntt(vl &a) {
9     int n = SZ(a), L = 31 - __builtin_clz(n);
10    static vl rt(2, 1);

```

```

8 for(static int k = 2, s = 2; k < n; k *=
9     2, s++) {
10    rt.resize(n);
11    ll z[] = {1, mod_pow(root, mod >> s, mod
12        )};
13    FOR(i, k, 2 * k) rt[i] = rt[i / 2] * z[i
14        & 1] % mod;
15    }
16    vi rev(n);
17    REP(i, n) rev[i] = (rev[i / 2] | (i & 1)
18        << L) / 2;
19    REP(i, n) if (i < rev[i]) swap(a[i], a[rev
20        [i]]);
21    for(int k = 1; k < n; k *= 2)
22        for(int i = 0; i < n; i += 2 * k) REP(j,
23            k) {
24                ll z = rt[j + k] * a[i + j + k] % mod,
25                &ai = a[i + j];
26                a[i + j + k] = ai - z + (z > ai ? mod
27                    : 0);
28                ai += (ai + z >= mod ? z - mod : z);
29            }
30    }
31    v1 conv(const vl &a, const vl &b) {
32        if(a.empty() || b.empty()) return {};
33        int s = SZ(a) + SZ(b) - 1, B = 32 -
34            __builtin_clz(s), n = 1 << B;
35        ll inv = mod_pow(n, mod - 2, mod);
36        vl L(a), R(b), out(n);
37        L.resize(n), R.resize(n);
38        ntt(L), ntt(R);
39        REP(i, n) out[i & (n - 1)] = inv * L[i] %
40            mod * R[i] % mod;
41        ntt(out);
42        return {out.begin(), out.begin() + s};
43    }

```

6.14 Numbers

- Bernoulli numbers

$$B_0 = 1, B_1^{\pm} = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

$$\sum_{j=0}^m \binom{m+1}{j} B_j = 0, \text{EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$$

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), S(n, 1) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

- Pentagonal number theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2} \right)$$

- Catalan numbers

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$$

$$C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$$

- Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

6.15 Pisano number

```

1 // pisano number: 費氏數列 mod m
2 // 情況下多長會循環
3 // Can be proved under O(6m)
4 ll find_pisano(ll m) {
5     ll a = 0, b = 1, c;
6     for(i=0;;i++) {
7         c = (a+b) % m;
8         a = b; b = c;
9         if(!a && b == 1)
10             return i+1;
11     }
12 }

```

6.16 Pollard-Rho

```

1 #define ull unsigned long long
2 #define ldb long double
3
4 vector<ll> factor;
5 vector<pair<ll,ll>> fac;
6
7 ll fpow(ll x, ll y, ll p) {
8     ll res = 1;
9     while (y) {
10         if (y & 1) res = (__int128)res * x % p;
11         x = (__int128)x * x % p;
12         y >>= 1;
13     }
14     return res;
15 }
16
17 bool mr(ll x, ll p) {

```

```

18 if (fpow(x, p - 1, p) != 1) return 0;
19 ll y = p - 1, z;
20 while (!(y & 1)) {
21     y >>= 1;
22     z = fpow(x, y, p);
23     if (z != 1 && z != p - 1) return 0;
24     if (z == p - 1) return 1;
25 }
26 return 1;
27 }
28
29 // Miller Rabin ~O(Log p)
30 bool is_prime(ll p) {
31     if (p < 2) return 1;
32     if (p==2 || p==3 || p==5 || p==7 || p==43)
33         return 1;
34     return mr(2,p) && mr(3,p) && mr(5,p) && mr
35         (7,p) && mr(43,p);
36 }
37 // O(1) 快速乘(防LL overflow)
38 ll ksc(u11 x, u11 y, ll p) {
39     return (x*y-(u11)((ldb)x/p*y)*p+p)%p;
40 }
41 //求n任一真因数(需保证n非质数) O(n^1/4)
42 ll pollar_rho(ll n) {
43     ll x,y,z,c,g,i,j;
44     while(1) {
45         x = y = rand()%n;
46         z = 1;
47         c = rand()%n;
48         i = 0, j = 1;
49         while(++i) {
50             x = (ksc(x,x,n) + c)%n;
51             z = ksc(z,abs(y-x),n);
52             if(x == y || !z) break;
53             if(!(i%127) || i == j) {
54                 g = __gcd(z,n);
55                 if(g > 1) return g;
56                 if(i == j) y = x, j <= 1;
57             }
58         }
59     }
60 }
61
62 void factorization(ll n) {
63     while(!is_prime(n)) {
64         ll f = pollar_rho(n);
65         while(!is_prime(f)) {
66             f = pollar_rho(f);
67         }
68         ll cou = 0;
69         while(n%f == 0) n /= f, cou++;
70         fac.push_back({f,cou});
71     }
72     if(n != 1) fac.push_back({n,1});
73 }
74
75 void get_factors(ll now, ll cou) {
76     if(now >= fac.size()) {
77         factor.push_back(cou);
78         return;
79     }
80     get_factors(now+1,cou);

```

```

81 for(ll i=1;i<=fac[now].second;i++) {
82     cou *= fac[now].first;
83     get_factors(now+1,cou);
84 }
85 }

```

6.17 Primes

```

1 /* 12721 13331 14341 75577 123457 222557
556679 999983 1097774749 1076767633
100102021 999997771 1001010013
1000512343 987654361 999991231 999888733
98789101 987777733 999991921 1010101333
1010102101 1000000000039
100000000000037 2305843009213693951
4611686018427387847 9223372036854775783
18446744073709551557 */

```

6.18 Theorem

- Modular Arithmetic

$$(a + b) \bmod m = (a \bmod m + b \bmod m) \bmod m$$

$$(a - b) \bmod m = (a \bmod m - b \bmod m) \bmod m$$

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

$$a^b \bmod m = (a \bmod m)^{b \bmod m-1} \bmod m$$

- Cramer's rule

$$\begin{aligned} ax + by &= e & \Rightarrow & \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ cx + dy &= f & \Rightarrow & \begin{aligned} y &= \frac{af - ec}{ad - bc} \end{aligned} \end{aligned}$$

- Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

- Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

- Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ spanning trees.
- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

- Erdős-Gallai theorem

A sequence of nonnegative integers $d_1 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \dots + d_n$ is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every $1 \leq k \leq n$.

- Gale-Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Fulkerson-Chen-Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \dots \geq a_n$ is digraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

- Möbius inversion formula

$$\begin{aligned} f(n) &= \sum_{d|n} g(d) & \Leftrightarrow & g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) \\ f(n) &= \sum_{n|d} g(d) & \Leftrightarrow & g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d) \end{aligned}$$

- Spherical cap

- A portion of a sphere cut off by a plane.
- r : sphere radius, a : radius of the base of the cap, h : height of the cap, θ : $\arcsin(a/r)$.
- Volume $= \frac{\pi h^2}{6} (3r - h) / 3 = \frac{\pi h}{6} (3a^2 + h^2) / 6 = \frac{\pi r^3}{6} (2 + \cos \theta) (1 - \cos \theta)^2 / 3$.
- Area $= 2\pi r h = \pi(a^2 + h^2) = 2\pi r^2 (1 - \cos \theta)$.

6.19 Triangle

```

1 // Counts x, y >= 0 such that Ax + By <= C.
   Requires A, B > 0. Runs in log time.
2 // Also representable as sum_{0 <= x <= C /
   A} floor((C - Ax) / B + 1).
3 ll count_triangle(ll A, ll B, ll C) {
4     if(C < 0) return 0;
5     if(A < B) swap(A, B);
6     ll m = C / A, k = A / B;
7     ll h = (C - m * A) / B + 1;
8     return m * (m + 1) / 2 * k + (m + 1) * h
9         + count_triangle(B, A - k * B, C -
10             B * (k * m + h));
11 }
12
13 // Counts 0 <= x < RA, 0 <= y < RB such that
   Ax + By <= C. Requires A, B > 0.
14 ll count_triangle_rectangle_intersection(ll
   A, ll B, ll C, ll RA, ll RB) {
15     if(C < 0 || RA <= 0 || RB <= 0) return
16         0;
17     if(C < A * (RA - 1) + B * (RB - 1))
18         return RA * RB;
19     return count_triangle(A, B, C) -
20         count_triangle(A, B, C - A * RA) -
21         count_triangle(A, B, C - B * RB);
22 }

```

6.20 Xor-Basis

```

1 template<int B>
2 struct xor_basis {
3     using T = long long;
4     bool zero = false, change = false;
5     int cnt = 0;
6     array<T, B> p = {};
7     vector<T> d;
8     void insert(T x) {
9         IREP(i, B) {
10             if(x >> i & 1) {
11                 if(!p[i]) {
12                     p[i] = x, cnt++;
13                     change = true;
14                     return;
15                 } else x ^= p[i];
16             }
17         }
18         if(!zero) zero = change = true;
19     }
20     T get_min() {
21         if(zero) return 0;
22         REP(i, B) if(p[i]) return p[i];
23     }
24     T get_max() {
25         T ans = 0;
26         IREP(i, B) ans = max(ans, ans ^ p[i]);
27         return ans;
28     }
29     T get_kth(long long k) {
30         k++;
31         if(k == 1 && zero) return 0;

```



```

32 k -= zero;
33 if(k >= (1LL << cnt)) return -1;
34 update();
35 T ans = 0;
36 REP(i, SZ(d)) if(k >> i & 1) ans ^= d[i];
37 return ans;
38 }
39 bool contains(T x) {
40     if(x == 0) return zero;
41     IREP(i, B) if(x >> i & 1) x ^= p[i];
42     return x == 0;
43 }
44 void merge(const xor_basis& other) { REP(i, B) if(other.p[i]) insert(other.p[i]); }
45 void update() {
46     if(!change) return;
47     change = false;
48     d.clear();
49     REP(j, B) IREP(i, j) if(p[j] >> i & 1) p[j] ^= p[i];
50     REP(i, B) if(p[i]) d.pb(p[i]);
51 }
52 };

```

6.21 找實根

```

1 //  $an^x + \dots + a_1x + a_0 = 0$ ;
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 }

```

7 Square root decomposition

7.1 MoAlgo

```

1 struct qry{
2     int ql,qr,id;
3 };
4 template<class T>struct Mo{
5     int n,m;
6     vector<pii>ans;
7     Mo(int _n,int _m): n(_n),m(_m){
8         ans.resize(m);
9     }
10    void solve(vector<T>&v,vector<qry>&q){
11        int l = 0, r = -1;
12        vector<int>cnt,cntcnt;
13        cnt.resize(n+5);
14        cntcnt.resize(n+5);
15        int mx = 0;
16        function<void(int)>add = [&](int pos){
17            cntcnt[cnt[v[pos]]]--;
18            cnt[v[pos]]++;
19            cntcnt[cnt[v[pos]]]++;
20            mx = max(mx,cnt[v[pos]]);
21        };
22        function<void(int)>sub = [&](int pos){
23            if(!--cntcnt[cnt[v[pos]]] and cnt[v[pos]]==mx)mx--;
24            cnt[v[pos]]--;
25            cntcnt[cnt[v[pos]]]++;
26            mx = max(mx,cnt[v[pos]]);
27        };
28        sort(all(q), [&](qry a, qry b){
29            static int B = max((int)1, n/max((int)sqrt(m), (int)1));
30            if(a.ql/B != b.ql/B) return a.ql < b.ql;
31            if((a.ql/B)&1) return a.qr > b.qr;

```

```

32         return a.qr < b.qr;
33     });
34     for(auto [ql,qr,id]:q){
35         while(l>ql)add(--l);
36         while(r<qr)add(++r);
37         while(l<ql)sub(l++);
38         while(r>qr)sub(r--);
39         ans[id] = {mx,cntcnt[mx]};
40     }
41 }
42 };

```

7.2 Mos Algorithm On Tree

```

1 /*
2 Mo's Algorithm On Tree
3 Preprocess:
4 1) LCA
5 2) dfs with in[u] = dft++, out[u] = dft++
6 3) ord[in[u]] = ord[out[u]] = u
7 4) bitset<MAXN> inset
8 */
9 struct Query {
10     int L, R, LBid, lca;
11     Query(int u, int v) {
12         int c = LCA(u, v);
13         if (c == u || c == v)
14             q.lca = -1, q.L = out[c ^ u ^ v], q.R = out[c];
15         else if (out[u] < in[v])
16             q.lca = c, q.L = out[u], q.R = in[v];
17         else
18             q.lca = c, q.L = out[v], q.R = in[u];
19         q.Lid = q.L / blk;
20     }
21     bool operator<(const Query &q) const {
22         if (LBid != q.LBid) return LBid < q.LBid;
23         return R < q.R;
24     }
25 };
26 void flip(int x) {
27     if (inset[x]) sub(arr[x]); // TODO
28     else add(arr[x]); // TODO
29     inset[x] = ~inset[x];
30 }
31 void solve(vector<Query> query) {
32     sort(ALL(query));
33     int L = 0, R = 0;
34     for (auto q : query) {
35         while (R < q.R) flip(ord[++R]);
36         while (L > q.L) flip(ord[--L]);
37         while (R > q.R) flip(ord[R--]);
38         while (L < q.L) flip(ord[L++]);
39         if (~q.lca) add(arr[q.lca]);
40         // answer query
41         if (~q.lca) sub(arr[q.lca]);
42     }
43 }

```

7.3 分塊 cf455D

```

1 const ll block_siz = 320;
2 const ll maxn = 100005;
3 ll a[maxn];
4 ll cnt[block_siz+1][maxn]; // i-th block, k'
5 s cou
6 deque<ll> q[block_siz+1];
7
8 void print_all(ll n)
9 {
10     for(int i=0;i<n;i++)
11     {
12         cout << q[i/block_siz][i-i/block_siz*block_siz] << ' ';
13     }
14     cout << endl << endl;
15 }
16
17 int main()
18 {
19     Crbubble
20     ll n,m,i,k,t;
21     ll l,r,ord,pre,id,id2, ans = 0;
22     cin >> n;
23     for(i=0;i<n;i++)
24     {
25         cin >> a[i];
26         id = i/block_siz;
27         q[id].push_back(a[i]);
28         cnt[id][a[i]]++;
29     }
30     cin >> t;
31     while(t--)
32     {
33         cin >> ord >> l >> r;
34         l = (l+ans-1)%n+1 -1;
35         r = (r+ans-1)%n+1 -1;
36         if(l > r) swap(l,r);
37         id = l/block_siz; l %= block_siz;
38         id2 = r/block_siz; r %= block_siz;
39         if(ord == 1)
40         {
41             if(id == id2)
42             {
43                 pre = q[id][r];
44                 for(i=r;i>l;i--)
45                 {
46                     q[id][i] = q[id][i-1];
47                 }
48                 q[id][l] = pre;
49             }
50             else
51             {
52                 pre = q[id].back();
53                 cnt[id][pre]--;
54                 q[id].pop_back();
55                 for(i=id+1;i<id2;i++)
56                 {
57                     q[i].push_front(pre);
58                     cnt[i][pre]++;
59                     pre = q[i].back();
60                     cnt[i][pre]--;
61                     q[i].pop_back();

```

```

62     q[id2].push_front(pre);
63     cnt[id2][pre]++;
64     pre = q[id2][r+1];
65     cnt[id2][pre]--;
66     q[id2].erase(q[id2].begin()+
67         r+1);
68     q[id].insert(q[id].begin()+1
69         , pre);
70     cnt[id][pre]++;
71     //print_all(n);
72 }
73 else
74 { // query m cnt
75     cin >> m;
76     m = (m+ans-1)%n+1;
77     ans = 0;
78     if(id == id2)
79     {
80         for(i=1;i<=r;i++) ans += (q[
81             id][i] == m);
82     }
83     else
84     {
85         for(i=1;i<block_size;i++) ans
86             += (q[id][i] == m);
87         for(i=0;i<=r;i++) ans += (q[
88             id2][i] == m);
89         for(i=id+1;i<id2;i++) ans +=
90             cnt[i][m];
91     }
92     cout << ans << endl;
93 }
94 return 0;
95 }

```

8 Tree

8.1 centroidDecomposition

```

1 vector<vector<int>>>g;
2 vector<int>sz,tmp;
3 vector<bool>vis; //visit_centroid
4 int tree_centroid(int u,int n){
5     function<void(int,int)>dfs1 = [&](int u,
6         int p){
7         sz[u] = 1;
8         for(auto v:g[u]){
9             if(v==p)continue;
10            if(vis[v])continue;
11            dfs1(v,u);
12            sz[u]+=sz[v];
13        }
14    };
15    function<int(int,int)>dfs2 = [&](int u,int
16        p){
17        for(auto v:g[u]){
18            if(v==p)continue;
19            if(vis[v])continue;
20            if(sz[v]*2<n)continue;
21            return dfs2(v,u);
22        }
23        return u;
24    };
25    dfs1(u,-1);
26    return dfs2(u,-1);
27 }

```

```

18 sort(queries.begin(), queries.end());
19
20 // TODO: initialize data structure
21
22 int cur_l = 0;
23 int cur_r = -1;
24 // invariant: data structure will always
25 // reflect the range [cur_l, cur_r]
26 for (Query q : queries) {
27     while (cur_l > q.l) {
28         cur_l--;
29         add(cur_l);
30     }
31     while (cur_r < q.r) {
32         cur_r++;
33         add(cur_r);
34     }
35     while (cur_l < q.l) {
36         remove(cur_l);
37         cur_l++;
38     }
39     while (cur_r > q.r) {
40         remove(cur_r);
41         cur_r--;
42     }
43     answers[q.idx] = get_answer();
44 }
45 return answers;
46 }

```

```

26 int cal(int u,int p = -1,int deep = 1){
27     int ans = 0;
28     tmp.pb(deep);
29     sz[u] = 1;
30     for(auto v:g[u]){
31         if(v==p)continue;
32         if(vis[v])continue;
33         ans+=cal(v,u,deep+1);
34         sz[u]+=sz[v];
35     }
36     //calculate the answer
37     return ans;
38 }
39 int centroid_decomposition(int u,int
40     tree_size){
41     int center = tree_centroid(u,tree_size);
42     vis[center] = 1;
43     int ans = 0;
44     for(auto v:g[center]){
45         if(vis[v])continue;
46         ans+=cal(v);
47         for(int i = sz(tmp)-sz[v];i<sz(tmp);++i)
48             //update
49     }
50     while(!tmp.empty()){
51         //roll_back(tmp.back())
52         tmp.pop_back();
53     }
54     for(auto v:g[center]){
55         if(vis[v])continue;
56         ans+=centroid_decomposition(v,sz[v]);
57     }
58     return ans;
59 }

```

8.2 HeavyLight

```

1 #include<vector>
2 #define MAXN 100005
3 int siz[MAXN],max_son[MAXN],pa[MAXN],dep[
4     MAXN];
5 int link_top[MAXN],link[MAXN],cnt;
6 vector<int> G[MAXN];
7 void find_max_son(int u){
8     siz[u]=1;
9     max_son[u]=-1;
10    for(auto v:G[u]){
11        if(v==pa[u])continue;
12        pa[v]=u;
13        dep[v]=dep[u]+1;
14        find_max_son(v);
15        if(max_son[u]==-1||siz[v]>siz[max_son[
16            u]])max_son[u]=v;
17        siz[u]+=siz[v];
18    }
19 }
20 void build_link(int u,int top){
21     link[u]=++cnt;
22     link_top[u]=top;
23     if(max_son[u]==-1)return;
24     build_link(max_son[u],top);
25 }

```

```

23 for(auto v:G[u]){
24     if(v==max_son[u]||v==pa[u])continue;
25     build_link(v,v);
26 }
27 int find_lca(int a,int b){
28     //求LCA · 可以在過程中對區間進行處理
29     int ta=link_top[a],tb=link_top[b];
30     while(ta!=tb){
31         if(dep[ta]<dep[tb]){
32             swap(ta,tb);
33             swap(a,b);
34         }
35         //這裡可以對a所在的鏈做區間處理
36         //區間為(Link[ta],Link[a])
37         ta=link_top[a=pa[ta]];
38     }
39     //最後a,b會在同一條鏈 · 若a!=b還要在進行一
40     //次區間處理
41     return dep[a]<dep[b]?a:b;
42 }

```

8.3 HLD

```

1 struct heavy_light_decomposition{
2     int n;
3     vector<int>dep,father,sz,mxson,topf,id;
4     vector<vector<int>>>g;
5     heavy_light_decomposition(int _n = 0) : n(
6         _n) {
7         g.resize(n+5);
8         dep.resize(n+5);
9         father.resize(n+5);
10        sz.resize(n+5);
11        mxson.resize(n+5);
12        topf.resize(n+5);
13        id.resize(n+5);
14    }
15    void add_edge(int u, int v){
16        g[u].push_back(v);
17        g[v].push_back(u);
18    }
19    void dfs(int u,int p){
20        dep[u] = dep[p]+1;
21        father[u] = p;
22        sz[u] = 1;
23        mxson[u] = 0;
24        for(auto v:g[u]){
25            if(v==p)continue;
26            dfs(v,u);
27            sz[u]+=sz[v];
28            if(sz[v]>sz[mxson[u]])mxson[u] = v;
29        }
30    }
31    void dfs2(int u,int top){
32        static int idn = 0;
33        topf[u] = top;
34        id[u] = ++idn;
35        if(mxson[u])dfs2(mxson[u],top);
36        for(auto v:g[u]){
37            if(v!=father[u] and v!=mxson[u]){
38                dfs2(v,v);
39            }
40        }
41    }
42 }

```

7.4 莫隊

```

1 void remove(idx); // TODO: remove value at
2   idx from data structure
3 void add(idx); // TODO: add value at idx
4   from data structure
5 int get_answer(); // TODO: extract the
6   current answer of the data structure
7
8 int block_size;
9
10 struct Query {
11     int l, r, idx;
12     bool operator<(Query other) const
13     {
14         return make_pair(l / block_size, r)
15             <
16             make_pair(other.l /
17                 block_size, other.r);
18     }
19 };
20
21 vector<int> mo_s_algorithm(vector<Query>
22     queries) {
23     vector<int> answers(queries.size());
24 }

```

```

38     }
39   }
40 }
41 void build(int root){
42   dfs(root,0);
43   dfs2(root,root);
44 }
45 vector<pair<int, int>> path(int u,int v){
46   vector<pair<int, int>>ans;
47   while(topf[u]!=topf[v]){
48     if(dep[topf[u]<dep[topf[v]])swap(u,v)
49     ;
50     ans.push_back({id[topf[u]], id[u]});
51     u = father[topf[u]];
52   }
53   if(id[u]>id[v])swap(u,v);
54   ans.push_back({id[u], id[v]});
55   return ans;
56 };

```

```

39 int cnt; /*時間戳*/
40 vector<int> G[MAXN+5];
41 void dfs(int x,int pa){
42   in[x]=++cnt;
43   vs[cnt]=x;
44   for(VIT i=G[x].begin();i!=G[x].end();++i){
45     if(*i==pa)continue;
46     dep[*i]=dep[x]+1;
47     dfs(*i,x);
48     vs[++cnt]=x;
49   }
50 }
51
52 inline int find_lca(int a,int b){
53   if(in[a]>in[b])swap(a,b);
54   return RMQ(in[a],in[b]);
55 }

```

8.5 link cut tree

8.4 LCA

```

1 const int MAXN=200000; // 1-base
2 const int MLG=__lg(MAXN) + 1; //Log2(MAXN)
3   +1;
4 int pa[MLG+2][MAXN+5];
5 int dep[MAXN+5];
6 vector<int> G[MAXN+5];
7 void dfs(int x,int p=0){ //dfs(root);
8   pa[0][x]=p;
9   for(int i=0;i<=MLG;++i)
10     pa[i+1][x]=pa[i][pa[i][x]];
11   for(auto &i:G[x]){
12     if(i==p)continue;
13     dep[i]=dep[x]+1;
14     dfs(i,x);
15   }
16 inline int jump(int x,int d){
17   for(int i=0;i<=MLG;++i)
18     if((d>>i)&1) x=pa[i][x];
19   return x;
20 }
21 inline int find_lca(int a,int b){
22   if(dep[a]>dep[b])swap(a,b);
23   b=jump(b,dep[b]-dep[a]);
24   if(a==b)return a;
25   for(int i=MLG;i>0;--i){
26     if(pa[i][a]!=pa[i][b]){
27       a=pa[i][a];
28       b=pa[i][b];
29     }
30   }
31   return pa[0][a];
32 }
33 //用樹壓平做
34 #define MAXN 100000
35 typedef vector<int>::iterator VIT;
36 int dep[MAXN+5], in[MAXN+5];
37 int vs[2*MAXN+5];

```

```

1 struct splay_tree{
2   int ch[2],pa; //子節點跟父母
3   bool rev; //反轉的懶惰標記
4   splay_tree():pa(0),rev(0){ch[0]=ch[1]=0;}
5 };
6 vector<splay_tree> nd;
7 //有的時候用vector會MLE，要注意
8 //這邊以node[0]作為null節點
9 bool isroot(int x){ //判斷是否為這棵splay
10   tree的根
11   return nd[nd[x].pa].ch[0]!=x&&nd[nd[x].pa]
12     .ch[1]!=x;
13 }
14 void down(int x){ //懶惰標記下推
15   if(nd[x].rev){
16     if(nd[x].ch[0]nd[nd[x].ch[0]].rev^=1;
17     if(nd[x].ch[1]nd[nd[x].ch[1]].rev^=1;
18     swap(nd[x].ch[0],nd[x].ch[1]);
19     nd[x].rev=0;
20   }
21 void push_down(int x){ //所有祖先懶惰標記下推
22   if(!isroot(x))push_down(nd[x].pa);
23   down(x);
24 }
25 void up(int x){ //將子節點的資訊向上更新
26 void rotate(int x){ //旋轉，會自行判斷轉的方向
27   int y=nd[x].pa,z=nd[y].pa,d=(nd[y].ch[1]==
28     x);
29   nd[x].pa=z;
30   if(!isroot(y))nd[z].ch[nd[z].ch[1]==y]=x;
31   nd[y].ch[d]=nd[x].ch[d^1];
32   nd[nd[y].ch[d]].pa=y;
33   nd[y].pa=x,nd[x].ch[d^1]=y;
34   up(y),up(x);
35 }
36 void splay(int x){ //將x伸展到splay tree的根
37   push_down(x);
38   while(!isroot(x)){
39     int y=nd[x].pa;

```

```

38   if(!isroot(y)){
39     int z=nd[y].pa;
40     if((nd[z].ch[0]==y)^(nd[y].ch[0]==x))
41       rotate(y);
42     else rotate(x);
43   }
44   rotate(x);
45 }
46 int access(int x){
47   int last=0;
48   while(x){
49     splay(x);
50     nd[x].ch[1]=last;
51     up(x);
52     last=x;
53     x=nd[x].pa;
54   }
55   return last; //access後splay tree的根
56 }
57 void access(int x,bool is=0){ //is=0就是一般
58   的access
59   int last=0;
60   while(x){
61     splay(x);
62     if(is&&nd[nd[x].pa]{
63       //printf("%d\n",max(nd[last].ma,nd[nd[
64         x].ch[1]].ma));
65     }
66     nd[x].ch[1]=last;
67     up(x);
68     last=x;
69     x=nd[x].pa;
70   }
71 void query_edge(int u,int v){
72   access(u);
73   access(v,1);
74 }
75 void make_root(int x){
76   access(x),splay(x);
77   nd[x].rev^=1;
78 }
79 void make_root(int x){
80   nd[access(x)].rev^=1;
81   splay(x);
82 }
83 void cut(int x,int y){
84   make_root(x);
85   access(y);
86   splay(y);
87   nd[y].ch[0]=0;
88   nd[x].pa=0;
89 }
90 void cut_parents(int x){
91   access(x);
92   splay(x);
93   nd[nd[x].ch[0]].pa=0;
94   nd[x].ch[0]=0;
95 }
96 void link(int x,int y){
97   make_root(x);
98   nd[x].pa=y;
99 }
100 int find_root(int x){

```

```

100   x=access(x);
101   while(nd[x].ch[0])x=nd[x].ch[0];
102   splay(x);
103   return x;
104 }
105 int query(int u,int v){
106   //傳回uv路徑splay tree的根結點
107   //這種寫法無法求LCA
108   make_root(u);
109   return access(v);
110 }
111 int query_lca(int u,int v){
112   //假設求鏈上點權的總和，sum是子樹的權重和，
113   data是節點的權重
114   access(u);
115   int lca=access(v);
116   splay(u);
117   if(u==lca){
118     //return nd[lca].data+nd[nd[lca].ch[1]].
119     sum
120   }else{
121     //return nd[lca].data+nd[nd[lca].ch[1]].
122     sum+nd[u].sum
123   }
124 }
125 struct EDGE{
126   int a,b,w;
127 }e[10005];
128 int n;
129 vector<pair<int,int>> G[10005];
130 //first表示子節點，second表示邊的編號
131 int pa[10005],edge_node[10005];
132 //pa是父母節點，暫存用的，edge_node是每個編
133 被存在哪個點裡面的陣列
134 void bfs(int root){
135   //在建構的時候把每個點都設成一個splay tree
136   queue<int> q;
137   for(int i=1;i<=n;++i)pa[i]=0;
138   q.push(root);
139   while(q.size()){
140     int u=q.front();
141     q.pop();
142     for(auto P:G[u]){
143       int v=P.first;
144       if(v!=pa[u]){
145         pa[v]=u;
146         nd[v].pa=u;
147         nd[v].data=e[P.second].w;
148         edge_node[P.second]=v;
149         up(v);
150         q.push(v);
151       }
152     }
153   }
154 }
155 void change(int x,int b){
156   splay(x);
157   //nd[x].data=b;
158   up(x);
159 }

```

8.6 Tree centroid

```

1 //找出其中一個樹重心
2 vector<int> size;
3
4 int ans = -1;
5 void dfs(int u, int parent = -1) {
6     size[u] = 1;
7     int max_son_size = 0;
8     for (auto v : Tree[u]) {
9         if (v == parent) continue;
10        dfs(v, u);
11        size[u] += size[v];
12        max_son_size = max(max_son_size, size[v]);
13    }
14    max_son_size = max(max_son_size, n - size[u]);
15    if (max_son_size <= n / 2) ans = u;
16 }

```

8.7 樹壓平

```

1 //紀錄in & out
2 vector<int> Arr;
3 vector<int> In, Out;
4 void dfs(int u) {
5     Arr.push_back(u);
6     In[u] = Arr.size() - 1;
7     for (auto v : Tree[u]) {
8         if (v == parent[u]) continue;
9         parent[v] = u;
10        dfs(v);
11    }
12    Out[u] = Arr.size() - 1;
13 }
14
15 //進去出來都紀錄
16 vector<int> Arr;
17 void dfs(int u) {
18     Arr.push_back(u);
19     for (auto v : Tree[u]) {
20         if (v == parent[u]) continue;
21         parent[v] = u;
22         dfs(v);
23     }
24     Arr.push_back(u);
25 }
26
27 //用Treap紀錄
28 Treap *root = nullptr;
29 vector<Treap *> In, Out;
30 void dfs(int u) {
31     In[u] = new Treap(cost[u]);
32     root = merge(root, In[u]);
33     for (auto v : Tree[u]) {
34         if (v == parent[u]) continue;
35         parent[v] = u;
36     }
37     dfs(v);
38     parent[v] = u;
39     dfs(v);
40 }
41 Out[u] = new Treap(0);
42 root = merge(root, Out[u]);
43 }
44 //Treap紀錄Parent
45 struct Treap {
46     Treap *lc = nullptr, *rc = nullptr;
47     Treap *pa = nullptr;
48     unsigned pri, size;
49     long long Val, Sum;
50     Treap(int Val):
51         pri(rand()), size(1),
52         Val(Val), Sum(Val) {}
53     void pull();
54 };
55
56 void Treap::pull() {
57     size = 1;
58     Sum = Val;
59     pa = nullptr;
60     if (lc) {
61         size += lc->size;
62         Sum += lc->Sum;
63         lc->pa = this;
64     }
65     if (rc) {
66         size += rc->size;
67         Sum += rc->Sum;
68         rc->pa = this;
69     }
70 }
71 //找出節點在中序的編號
72 size_t getIdx(Treap *x) {
73     assert(x);
74     size_t Idx = 0;
75     for (Treap *child = x->rc; x;) {
76         if (child == x->rc)
77             Idx += 1 + size(x->lc);
78         child = x;
79         x = x->pa;
80     }
81     return Idx;
82 }
83 //切出想要的東西
84 void move(Treap *&root, int a, int b) {
85     size_t a_in = getIdx(In[a]), a_out = getIdx(Out[a]);
86     auto [L, tmp] = splitK(root, a_in - 1);
87     auto [tree_a, R] = splitK(tmp, a_out - a_in + 1);
88     root = merge(L, R);
89     tie(L, R) = splitK(root, getIdx(In[b]));
90     root = merge(L, merge(tree_a, R));
91 }

```

9 string

9.1 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=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    }
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){

```

```

62        ans+=S[q[i]].cnt_dp*S[q[i]].ed;
63        if(~S[q[i]].fail) S[q[i]].fail.
64            cnt_dp+=S[q[i]].cnt_dp;
65    }
66    return ans;
67 }
68 /*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)*/
69 int match_1(const char *s) const {
70     int ans=0, id, p=0, t;
71     for(int i=0; s[i]; ++i){
72         id=s[i]-L;
73         while(!S[p].next[id]&&p=S[p].fail;
74             if(!S[p].next[id]) continue;
75             p=S[p].next[id];
76             if(S[p].ed) ans+=S[p].ed;
77             for(t=S[p].efl; ~t; t=S[t].efl){
78                 ans+=S[t].ed; /*因為都走efl邊所以保證匹配成功*/
79             }
80         }
81     }
82     return ans;
83 }
84 /*枚舉(s的子串串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     /*把數記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     }
105     return ans;
106 }
107 /*把AC自動機變成真的自動機*/
108 void evolution(){
109     for(qs=1; qs!=qe;){
110         int p=q[qs++];
111         for(int i=0; i<R-L; ++i)
112             if(S[p].next[i]==0) S[p].next[i]=S[S[p].fail].next[i];
113     }
114 }

```

9.2 De Bruijn sequence


```

1 constexpr int MAXC = 10, MAXN = 1e5 + 10;
2 struct DBSeq {
3     int C, N, K, L, buf[MAXC * MAXN]; // K <=
4         C^N
5     void dfs(int *out, int t, int p, int &ptr)
6     {
7         if (ptr >= L) return;
8         if (t > N) {
9             if (N % p) return;
10            for (int i = 1; i <= p && ptr < L; ++i)
11                out[ptr++] = buf[i];
12        } else {
13            buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
14            for (int j = buf[t - p] + 1; j < C; ++j)
15                buf[t] = j, dfs(out, t + 1, t, ptr);
16        }
17    }
18    void solve(int _c, int _n, int _k, int *
19        out) {
20        int p = 0;
21        C = _c, N = _n, K = _k, L = N + K - 1;
22        dfs(out, 1, 1, p);
23        if (p < L) fill(out + p, out + L, 0);
24    }
25 } dbs;

```

9.3 KMP

```

1 const int N = 1e6+5;
2 /*產生fail function*/
3 void kmp_fail(char *s, int len, int *fail){
4     int id=-1;
5     fail[0]=-1;
6     for(int i=1; i<len; ++i){
7         while(~id&&s[id+1]!=s[i]) id=fail[id];
8         if(s[id+1]==s[i]) ++id;
9         fail[i]=id;
10    }
11 }
12 vector<int> match_index;
13 /*以字串B匹配字串A，傳回匹配成功的數量(用B的
14    fail)*/
15 int kmp_match(char *A, int lenA, char *B, int
16    lenB, int *fail){
17     int id=-1, ans=0;
18     for(int i=0; i<lenA; ++i){
19         while(~id&&B[id+1]!=A[i]) id=fail[id];
20         if(B[id+1]==A[i]) ++id;
21         if(id==lenB-1){/*匹配成功*/
22             ++ans, id=fail[id];
23             match_index.emplace_back(i + 1 - lenB);
24         }
25     }
26     return ans;
27 }

```

9.4 manacher

```

1 //找最長迴文子字串
2 //原字串: asdsasdsa
3 //先把字串變成這樣: @#a#s#d#s#a#s#d#s#a#
4 void manacher(char *s, int len, int *z){
5     int l=0, r=0;
6     for(int i=1; i<len; ++i){
7         z[i]=r>i?min(z[2*i-1], 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    } //ans = max(z)-1
11 }

```

9.5 minimal string rotation

```

1 //找最小循環表示法起始位置
2 int min_string_rotation(const string &s){
3     int n=s.size(), i=0, j=1, k=0;
4     while(i<n&&j<n&&k<n){
5         int t=s[(i+k)%n]-s[(j+k)%n];
6         ++k;
7         if(t){
8             if(t>0) i+=k;
9             else j+=k;
10            if(i==j) ++j;
11            k=0;
12        }
13    }
14    return min(i, j); //最小循環表示法起始位置
15 }

```

9.6 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])]++;
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    }
26 }

```

```

25     i = first[int(c)] + ranks[i];
26 } while (i != begin);
27 return res;
28 }

```

9.7 Rolling Hash

```

1 const ll M = 911382323, mod = 972663749;
2 ll Get(vector<ll> &h, int l, int r) {
3     if(!l) return h[r]; // p[i] = M^i % mod
4     ll ans = (h[r] - h[l - 1] * p[r - l + 1])
5         % mod;
6     return (ans + mod) % mod;
7 }
8 vector<ll> Hash(string s) {
9     vector<ll> ans(SZ(s));
10    ans[0] = s[0];
11    for(int i = 1; i < SZ(s); ++i) ans[i] = (
12        ans[i - 1] * M + s[i]) % mod;
13    return ans;
14 }

```

9.8 SAM

```

1 const int MAXM = 1000010;
2 struct SAM {
3     int tot, root, lst, mom[MAXM], mx[MAXM];
4     int nxt[MAXM][33], cnt[MAXM], in[MAXM];
5     int newNode() {
6         int res = ++tot;
7         fill(nxt[res], nxt[res] + 33, 0);
8         mom[res] = mx[res] = cnt[res] = in[res]
9             = 0;
10        return res;
11    }
12    void init() {
13        tot = 0;
14        root = newNode();
15        mom[root] = 0, mx[root] = 0;
16        lst = root;
17    }
18    void push(int c) {
19        int p = lst;
20        int np = newNode();
21        mx[np] = mx[p] + 1;
22        for (; p && nxt[p][c] == 0; p = mom[p])
23            nxt[p][c] = np;
24        if (p == 0) mom[np] = root;
25        else {
26            int q = nxt[p][c];
27            if (mx[p] + 1 == mx[q]) mom[np] = q;
28            else {
29                int nq = newNode();
30                mx[nq] = mx[p] + 1;
31                for (int i = 0; i < 33; ++i)
32                    nxt[nq][i] = nxt[p][i];
33                mom[nq] = mom[q];
34                mom[q] = nq;
35                mom[np] = nq;
36            }
37        }
38    }
39 }

```

```

35     for (; p && nxt[p][c] == q; p = mom[
36         p])
37         nxt[p][c] = nq;
38     }
39     lst = np, cnt[np] = 1;
40 }
41 void push(char *str) {
42     for (int i = 0; str[i]; ++i)
43         push(str[i] - 'a' + 1);
44 }
45 void count() {
46     for (int i = 1; i <= tot; ++i)
47         ++in[mom[i]];
48     queue<int> q;
49     for (int i = 1; i <= tot; ++i)
50         if (!in[i]) q.push(i);
51     while (!q.empty()) {
52         int u = q.front();
53         q.pop();
54         cnt[mom[u]] += cnt[u];
55         if (!in[mom[u]])
56             q.push(mom[u]);
57     }
58 }
59 } sam;

```

9.9 suffix array lcp

```

1 // Suffix Array: 將一字串所有後綴排序形成的
2 // array
3 // sa[i]: 第 i 大的後綴從哪開始 (0-index)
4 // rk[i]: 從 i 開始的後綴是第幾大
5 // lcp[i]: 第 i 和 i+1 個後綴的最長前綴長度
6 const int maxn = 100005; string s;
7 int sa[maxn], tmp[2][maxn], c[maxn];
8 void getSA() {
9     int *x=tmp[0], *y=tmp[1], m=256, n=s.size();
10    int i, k;
11    for(i=0; i<n; ++i) c[i] = 0;
12    for(i=0; i<n; ++i) c[x[i]] = s[i]++;
13    for(i=1; i<m; ++i) c[i] += c[i-1];
14    for(i=n-1; i>=0; i--) sa[--c[x[i]]] = i;
15    for(k=1; k<n; k<=1) {
16        for(i=0; i<m; ++i) c[i] = 0;
17        for(i=0; i<n; ++i) c[x[i]]++;
18        for(i=1; i<m; ++i) c[i] += c[i-1];
19        int p = 0;
20        for(i=n-k; i<n; ++i) y[p++] = i;
21        for(i=0; i<n; ++i)
22            if(sa[i] >= k) y[p++] = sa[i]-k;
23        for(i=n-1; i>=0; i--) sa[--c[x[y[i]]]] = y[i];
24        y[sa[0]] = p = 0;
25        for(i=1; i<n; ++i) {
26            int a = sa[i], b = sa[i-1];
27            if(x[a]==x[b] && a+k<n && b+k<n && x[a+k]==x[b+k]);
28            else p++;
29            y[sa[i]] = p;
30        }
31        if(n == p+1) break;
32    }
33 }

```

```

31 swap(x,y), m = p+1;
32 }
33 }
34 int rk[maxn], lcp[maxn];
35 void getLCP() {
36     int n = s.size(), val = 0, i;
37     for(i=0;i<n;i++) rk[sa[i]] = i;
38     for(i=0;i<n;i++) {
39         if(rk[i] == 0) lcp[rk[i]] = 0;
40         else {
41             if(val) --val;
42             int p = sa[rk[i]-1];
43             while(val+i<n && val+p<n && s[val+i]==
44                 s[val+p]) val++;
45             lcp[rk[i]] = val;
46         }
47     }

```

9.10 Trie

```

1 template<int ALPHABET = 26, char MIN_CHAR =
  'a'>
2 class trie {
3 public:
4     struct Node {
5         int go[ALPHABET];
6         Node() {
7             memset(go, -1, sizeof(go));
8         }
9     };
10
11     trie() {
12         newNode();
13     }
14
15     inline int next(int p, int v) {
16         return nodes[p].go[v] != -1 ? nodes[p].
17             go[v] : nodes[p].go[v] = newNode();
18     }
19
20     inline void insert(const vector<int>& a,
21         int p = 0) {
22         for(int v : a) {
23             p = next(p, v);
24         }
25
26         inline void clear() {
27             nodes.clear();
28             newNode();
29         }
30
31         inline int longest_common_prefix(const
32             vector<int>& a, int p = 0) const {
33             int ans = 0;
34             for(int v : a) {
35                 if(nodes[p].go[v] != -1) {
36                     ans += 1;
37                     p = nodes[p].go[v];
38                 } else {
39                     break;
40                 }
41             }

```

```

39     }
40     return ans;
41 }
42
43 private:
44     vector<Node> nodes;
45
46     inline int newNode() {
47         nodes.emplace_back();
48         return (int) nodes.size() - 1;
49     }
50 };

```

9.11 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];
9         if(i+z[i]-1>r)r=i+z[i]-1,l=i;
10    }

```

10 tools

10.1 bitset

```

1 bitset<size> b(a):長度為size · 初始化為a
2 b[i]:第i位元的值(0 or 1)
3 b.size():有幾個位元
4 b.count():有幾個1
5 b.set():所有位元設為1
6 b.reset():所有位元設為0
7 b.flip():所有位元反轉

```

10.2 Counting Sort

```

1 vector<unsigned> counting_sort(const vector<
  unsigned> &Arr, unsigned K) {
2     vector<unsigned> Bucket(k, 0);
3     for(auto x: Arr)
4         ++Bucket[x];
5     partial_sum(Bucket.begin(), Bucket.end(),
6         Bucket.begin());
7     vector<unsigned> Ans(Arr.size());
8     for(auto Iter = Arr.rbegin(); Iter != Arr.
9         rend(); ++Iter) Ans[--Bucket[*Iter]] =
10         *Iter;
11     return Ans;
12 }

```

10.3 DuiPai

```

1 int main(){
2     string sol,bf,make;
3     cout<<"Your solution file name :";
4     cin>>sol;
5     cout<<"Brute force file name :";
6     cin>>bf;
7     cout<<"Make data file name :";
8     cin>>make;
9     system(("g++ "+sol+" -o sol").c_str());
10    system(("g++ "+bf+" -o bf").c_str());
11    system(("g++ "+make+" -o make").c_str());
12    for(int t = 0; t<10000; ++t){
13        system("./make > ./1.in");
14        double st = clock();
15        system("./sol < ./1.in > ./1.ans");
16        double et = clock();
17        system("./bf < ./1.in > ./1.out");
18        if(system("diff ./1.out ./1.ans")) {
19            printf("\033[0;31mWrong Answer\033[0m
20                on test #%d", t);
21            return 0;
22        }
23        else if(et-st>=2000){
24            printf("\033[0;32mTime Limit exceeded
25                \033[0m on test #%d, Time %.0lfms\
26                n", t, et-st);
27            return 0;
28        }
29        else {
30            printf("\033[0;32mAccepted\033[0m
31                m on test #%d, Time %.0lfms\
32                n", t, et - st);
33        }
34    }
35 }

```

10.4 relabel

```

1 template<class T>
2 vector<int> Discrete(const vector<T>&v){
3     vector<int>ans;
4     vector<T>tmp(v);
5     sort(begin(tmp),end(tmp));
6     tmp.erase(unique(begin(tmp),end(tmp)),end(
7         tmp));
8     for(auto i:v)ans.push_back(lower_bound(
9         begin(tmp),end(tmp),i)-tmp.begin()+1);
10    return ans;
11 }

```

10.5 Template

```

1 #include <bits/extc++.h>
2 #include <bits/stdc++.h>
3 #pragma GCC optimize("O3,unroll-loops")
4 #pragma GCC target("avx2,bmi2,lzcnt,
5     popcnt")

```

```

5 #define IOS ios::sync_with_stdio(0),cin.tie
  (0),cout.tie(0)
6 #define int long long
7 #define double long double
8 #define pb push_back
9 #define sz(x) (int)(x).size()
10 #define all(v) begin(v),end(v)
11 #define debug(x) cerr<<#x<<" = "<<x<<'\n'
12 #define LINE cout<<"\n-----\n"
13 #define endl '\n'
14 #define VI vector<int>
15 #define F first
16 #define S second
17 #define MP(a,b) make_pair(a,b)
18 #define rep(i,m,n) for(int i = m;i<n;++i)
19 #define res(i,m,n) for(int i = m;i>n;--i)
20 #define gcd(a,b) __gcd(a,b)
21 #define lcm(a,b) a*b/gcd(a,b)
22 #define Case() int _;cin>>_;for(int Case =
23     1;Case<=_;++Case)
24 #define pii pair<int,int>
25 using namespace __gnu_cxx;
26 using namespace __gnu_pbds;
27 using namespace std;
28 template<typename K, typename cmp = less<K
29     >, typename T = thin_heap_tag> using
30     _heap = __gnu_pbds::priority_queue<K,
31     cmp, T>;
32 template<typename K, typename M = null_type
33     > using _hash = gp_hash_table<K, M>;
34 const int N = 1e6+5, L = 20, mod = 1e9+7;
35 const long long inf = 2e18+5;
36 const double eps = 1e-7, pi = acos(-1);
37 void solve(){
38 }
39 signed main(){
40     IOS;
41     solve();
42 }
43
44 //使用內建紅黑樹
45 template<class T, typename cmp=less<>>struct
46     _tree{//#include<bits/extc++.h>
47     tree<pair<T,int>,null_type,cmp,rb_tree_tag
48         ,tree_order_statistics_node_update>st;
49     int id = 0;
50     void insert(T x){st.insert({x,id++});}
51     void erase(T x){st.erase(st.lower_bound({x
52         ,0}));}
53     int order_of_key(T x){return st.
54         order_of_key(*st.lower_bound({x,0}));}
55     T find_by_order(int x){return st.
56         find_by_order(x)->first;}
57     T lower_bound(T x){return st.lower_bound({
58         x,0})->first;}
59     T upper_bound(T x){return st.upper_bound({
60         x,(int)1e9+7})->first;}
61     T smaller_bound(T x){return (--st.
62         lower_bound({x,0}))->first;}
63 };

```

10.6 TernarySearch

```

1 // return the maximum of $f(x)$ in $[L, r]$
2 double ternary_search(double l, double r) {
3     while(r - l > EPS) {
4         double m1 = l + (r - l) / 3;
5         double m2 = r - (r - l) / 3;
6         double f1 = f(m1), f2 = f(m2);
7         if(f1 < f2) l = m1;
8         else r = m2;
9     }
10    return f(l);
11 }
12
13 // return the maximum of $f(x)$ in $(L, r]$
14 int ternary_search(int l, int r) {
15     while(r - l > 1) {
16         int mid = (l + r) / 2;
17         if(f(m) > f(m + 1)) r = m;
18         else l = m;
19     }
20    return r;
21 }

```

10.7 time rand

```

1 #define st clock_t qua = clock();
2 #define ed cout << "time: " << (double)(
3     clock() - qua) / CLOCKS_PER_SEC << " sec\n";
4
5 unsigned int genseed() {
6     auto now = chrono::system_clock::now();
7     auto timestamp = chrono::duration_cast<
8         chrono::milliseconds>(now -
9         time_since_epoch());
10    return static_cast<unsigned int>(timestamp
11        .count());
12 }
13
14 int main() {
15     unsigned int seed = genseed();
16     mt19937 engine(seed);
17     cout << engine(); // random num
18     return 0;
19 }

```

10.8 TouristIO

```

1 static struct FastInput {
2     static constexpr int BUF_SIZE = 1 << 20;
3     char buf[BUF_SIZE];
4     size_t chars_read = 0;
5     size_t buf_pos = 0;
6     FILE *in = stdin;
7     char cur = 0;
8
9     inline char get_char() {
10        if(buf_pos >= chars_read) {

```

```

11        chars_read = fread(buf, 1, BUF_SIZE,
12            in);
13        buf_pos = 0;
14        buf[0] = (chars_read == 0 ? -1 : buf
15            [0]);
16    }
17    return cur = buf[buf_pos++];
18    // return cur = getchar_unlocked();
19 }
20
21 inline void tie(int) {}
22
23 inline explicit operator bool() {
24    return cur != -1;
25 }
26
27 inline static bool is_blank(char c) {
28    return c <= ' ';
29 }
30
31 inline bool skip_blanks() {
32    while(is_blank(cur) && cur != -1) {
33        get_char();
34    }
35    return cur != -1;
36 }
37
38 inline FastInput& operator>>(char& c) {
39    skip_blanks();
40    c = cur;
41    return *this;
42 }
43
44 inline FastInput& operator>>(string& s) {
45    if(skip_blanks()) {
46        s.clear();
47        do {
48            s += cur;
49        } while(!is_blank(get_char()));
50    }
51    return *this;
52 }
53
54 template<class T>
55 inline FastInput& read_integer(T& n) {
56    // unsafe, doesn't check that characters
57    // are actually digits
58    n = 0;
59    if(skip_blanks()) {
60        int sign = +1;
61        if(cur == '-') {
62            sign = -1;
63            get_char();
64        }
65        do {
66            n += n * (n << 3) + cur - '0';
67        } while(!is_blank(get_char()));
68    }
69    n *= sign;
70    return *this;
71 }
72
73 template<class T>
74 inline typename enable_if<is_integral<T>::
75     value, FastInput&>::type operator>>(T&
76     n) {

```

```

77    return read_integer(n);
78 }
79
80 #if!defined(_WIN32) || defined(_WIN64)
81 inline FastInput& operator>>(__int128& n)
82 {
83     return read_integer(n);
84 }
85 #endif
86
87 template<class T>
88 inline typename enable_if<
89     is_floating_point<T>::value, FastInput
90     &>::type operator>>(T& n) {
91     // not sure if really fast, for
92     // compatibility only
93     n = 0;
94     if(skip_blanks()) {
95         string s;
96         (*this) >> s;
97         sscanf(s.c_str(), "%Lf", &n);
98     }
99    return *this;
100 }
101 fast_input;
102
103 #define cin fast_input
104
105 static struct FastOutput {
106     static constexpr int BUF_SIZE = 1 << 20;
107     char buf[BUF_SIZE];
108     size_t buf_pos = 0;
109     static constexpr int TMP_SIZE = 1 << 20;
110     char tmp[TMP_SIZE];
111     FILE *out = stdout;
112
113     inline void put_char(char c) {
114         buf[buf_pos++] = c;
115         if(buf_pos == BUF_SIZE) {
116             fwrite(buf, 1, buf_pos, out);
117             buf_pos = 0;
118         }
119     }
120     // putchar_unlocked(c);
121 }
122
123 ~FastOutput() {
124     fwrite(buf, 1, buf_pos, out);
125 }
126
127 inline FastOutput& operator<<(char c) {
128    put_char(c);
129    return *this;
130 }
131
132 inline FastOutput& operator<<(const char*
133     s) {
134    while(*s) {
135        put_char(*s++);
136    }
137    return *this;
138 }
139
140 inline FastOutput& operator<<(const string
141     & s) {
142    for(int i = 0; i < (int) s.size(); i++)
143        {

```

```

144        put_char(s[i]);
145    }
146    return *this;
147 }
148
149 template<class T>
150 inline char* integer_to_string(T n) {
151     // beware of TMP_SIZE
152     char* p = tmp + TMP_SIZE - 1;
153     if(n == 0) {
154         *--p = '0';
155     } else {
156         bool is_negative = false;
157         if(n < 0) {
158             is_negative = true;
159             n = -n;
160         }
161         while(n > 0) {
162             *--p = (char) ('0' + n % 10);
163             n /= 10;
164         }
165         if(is_negative) {
166             *--p = '-';
167         }
168     }
169    return p;
170 }
171
172 template<class T>
173 inline typename enable_if<is_integral<T>::
174     value, char*>::type stringify(T n) {
175    return integer_to_string(n);
176 }
177
178 #if!defined(_WIN32) || defined(_WIN64)
179 inline char* stringify(__int128 n) {
180    return integer_to_string(n);
181 }
182 #endif
183
184 template<class T>
185 inline typename enable_if<
186     is_floating_point<T>::value, char*>::
187     type stringify(T n) {
188    sprintf(tmp, "%.17f", n);
189    return tmp;
190 }
191
192 template<class T>
193 inline FastOutput& operator<<(const T& n)
194 {
195     {
196         auto p = stringify(n);
197         for(; *p != 0; p++) {
198             put_char(*p);
199         }
200     }
201    return *this;
202 }
203 fast_output;
204
205 #define cout fast_output

```

ACM ICPC Team Reference - DreaminBubble

Contents

1	Computational Geometry	1
1.1	Geometry	1
1.2	MinCircleCover	2
1.3	最近點對	3
2	DP	3
2.1	basic DP	3
2.2	DP on Graph	3
2.3	is planar	3
2.4	LineContainer	4
2.5	整體二分	4
2.6	斜率優化-動態凸包	5
2.7	斜率優化-單調隊列	5
3	Data Structure	5
3.1	2D BIT	5
3.2	BinaryTrie	5
3.3	BIT	6
3.4	DSU	6
3.5	DynamicMST	6

3.6	Dynamic Segment Tree	6
3.7	Kruskal	6
3.8	Lazytag Segment Tree	7
3.9	LiChaoST	7
3.10	pbds	7
3.11	Persistent DSU	7
3.12	Persistent Segment Tree	7
3.13	Prim	7
3.14	SegmentTree	7
3.15	sparse table	8
3.16	TimingSegmentTree	8
3.17	回滾並查集	8
3.18	掃描線 + 線段樹	8
3.19	陣列上 Treap	9
4	Flow	9
4.1	dinic	9
4.2	Gomory Hu	10
4.3	ISAP with cut	10
4.4	MinCostMaxFlow	10
4.5	Property	10
5	Graph	10
5.1	2-SAT	10
5.2	count-bridge-online	11
5.3	Dominator tree	11
5.4	manhattan-mst	11
5.5	Minimum Clique Cover	11
5.6	SCC	11
5.7	判斷環	12
5.8	最大團	12
5.9	枚舉極大團 Bron-Kerbosch	12

5.10	橋連通分量	12
5.11	雙連通分量 & 割點	12
6	Math	13
6.1	Basic	13
6.2	Bit Set	13
6.3	Combination	13
6.4	ExtendGCD	13
6.5	FFT	14
6.6	FWT	14
6.7	Gauss-Jordan	14
6.8	GCD-Convolution	14
6.9	InvGCD	14
6.10	LinearCongruence	14
6.11	Lucas	15
6.12	Matrix	15
6.13	NTT	15
6.14	Numbers	15
6.15	Pisano number	15
6.16	Pollard-Rho	15
6.17	Primes	16
6.18	Theorem	16
6.19	Triangle	16
6.20	Xor-Basis	16
6.21	找實根	17
7	Square root decomposition	17
7.1	MoAlgo	17
7.2	Mos Algorithm On Tree	17
7.3	分塊 cf455D	17
7.4	莫隊	18

8	Tree	18
8.1	centroidDecomposition	18
8.2	HeavyLight	18
8.3	HLD	18
8.4	LCA	19
8.5	link cut tree	19
8.6	Tree centroid	20
8.7	樹壓平	20
9	string	20
9.1	AC 自動機	20
9.2	De Bruijn sequence	20
9.3	KMP	21
9.4	manacher	21
9.5	minimal string rotation	21
9.6	reverseBWT	21
9.7	Rolling Hash	21
9.8	SAM	21
9.9	suffix array lcp	21
9.10	Trie	22
9.11	Z	22
10	tools	22
10.1	bitset	22
10.2	Counting Sort	22
10.3	DuiPai	22
10.4	relabel	22
10.5	Template	22
10.6	TenarySearch	23
10.7	time rand	23
10.8	TouristIO	23

ACM ICPC Judge Test - DreaminBubble

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
10    size_t block_size, bound;
11    void stack_size_dfs(size_t depth = 1) {
```

```
12        if (depth >= bound)
13            return;
14        int8_t ptr[block_size]; // 若無法編譯將
15            block_size 改成常數
16        memset(ptr, 'a', block_size);
17        cout << depth << endl;
18        stack_size_dfs(depth + 1);
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    double speed(int iter_num) {
27        const int block_size = 1024;
28        volatile int A[block_size];
29        auto begin = chrono::high_resolution_clock
30            ::now();
31        while (iter_num--)
32            for (int j = 0; j < block_size; ++j)
33                A[j] += j;
34        auto end = chrono::high_resolution_clock::
35            now();
36        chrono::duration<double> diff = end -
37            begin;
```

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

```
63    void runtime_error_5() {
64        // maybe illegal instruction
65        int a = 7122, b = 0;
66        cout << (a / b) << endl;
67    }
68    void runtime_error_6() {
69        // floating point exception
70        volatile int a = 7122, b = 0;
71        cout << (a / b) << endl;
72    }
73    void runtime_error_7() {
74        // call to abort.
75        assert(false);
76    }
77    } // namespace system_test
78
79    #include <sys/resource.h>
80    void print_stack_limit() { // only work in
81        Linux
82        struct rlimit l;
83        getrlimit(RLIMIT_STACK, &l);
84        cout << "stack_size = " << l.rlim_cur << "
85            byte" << endl;
86    }
87
```