

1 ComputationalGeometry

1.1 operators

```

1 typedef std::pair<double,double> Pt;
2 #define X first
3 #define Y second
4 const double eps = 1e-6;
5 Pt point( double x , double y ){
6     return make_pair( x , y );
7 }
8 Pt operator+( const Pt& p1 , const Pt& p2 ){
9     return Pt( p1.X + p2.X , p1.Y + p2.Y );
10 }
11 Pt operator-( const Pt& p1 , const Pt& p2 ){
12     return Pt( p1.X - p2.X , p1.Y - p2.Y );
13 }
14 double operator*( const Pt& p1 , const Pt& p2 ){
15     return p1.X * p2.X + p1.Y * p2.Y;
16 }
17 double operator^( const Pt& p1 , const Pt& p2 ){
18     return p1.X * p2.Y - p1.Y * p2.X;
19 }
20 Pt operator*( const Pt& p1 , const double& k ){
21     return Pt( p1.X * k , p1.Y * k );
22 }
23 Pt operator/( const Pt& p1 , const double& k ){
24     return Pt( p1.X / k , p1.Y / k );
25 }
26 bool equal( const double& a , const double& b ){
27     return b - eps < a && a < b + eps;
28 }
29 bool less( const double& a , const double& b ){
30     return a < b - eps;
31 }
32 bool lessOrEqual( const double& a , const double& b ){
33     return a < b + eps;
34 }
35 double abs( const Pt& p1 ){
36     return sqrt( p1 * p1 );
37 }
38 double area(){
39     double sum = 0;
40     for(int i = 0; i < n; i++) sum += 0.5*p[i]^i+1];
41     return sum;
42 }
43 Pt o;
44 D angle( const Pt& x ){
45     return atan2( x.Y , x.X );
46 }
47 bool cmp_angle( Pt a , Pt b ){
48     return angle( a - o ) < angle( b - o );
49 }
50 bool cmp_cross( Pt a , Pt b ){
51     return ( a - o ) ^ ( b - o ) > 0;
52 }

```

1.2 inter_par

```

1 int ori( const Pt& o , const Pt& a , const Pt& b ){

```

```

2     double cross = ( a - o ) ^ ( b - o );
3     if( fabs( cross ) < eps ) return 0;
4     return cross > 0 ? 1 : -1;
5 }
6 bool intersect(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){ //線段p1p2, p3p4
7     return ori(p1, p2, p3)^ori(p1, p2, p4) < 0;
8 }
9 int parallel(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){
10     return (p2-p1)^(p4-p3) == 0;
11 }
12 bool Collinear(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){ //共線
13     ori(p1, p2, p3) == 0;
14 }

```

2 DivideConquer

2.1 PojTree

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 typedef long long int ll;
5 typedef pair<int, ll> P;
6 #define idx first
7 #define w second
8
9 const int N = 10004;
10 const ll INF = (1ll << 60);
11
12 int vn;
13 ll k;
14 vector<P> graph[N];
15 vector<int> dist;
16 ll subtreeSz[N];
17 bool isCentroid[N];
18
19 void init()
20 {
21     for(int i = 1; i <= vn; i++)
22         graph[i].clear(), isCentroid[i] = false;
23 }
24
25 void buildTree()
26 {
27     for(int i = 1; i < vn; i++)
28     {
29         int u, v, l; scanf("%d %d %d", &u, &v, &l);
30         graph[u].push_back(P(v, l));
31         graph[v].push_back(P(u, l));
32     }
33 }
34
35 ll calSubsz(int v, int p)
36 {
37     subtreeSz[v] = 1;
38     for(auto c:graph[v])
39     {
40         if(isCentroid[c.idx] || c.idx == p) continue;
41         subtreeSz[v] += calSubsz(c.idx, v);
42     }

```

```

43     return subtreeSz[v];
44 }
45
46 P getCentroid(int v, int p, ll subsz)
47 {
48     P cen(-1, INF);
49     ll mxsonSz = -1;
50     for(auto c:graph[v])
51     {
52         if(c.idx == p || isCentroid[c.idx]) continue;
53         P res = getCentroid(c.idx, v, subsz);
54         if(res.w < cen.w) cen = res;
55         mxsonSz = max(mxsonSz, subtreeSz[c.idx]);
56     }
57     mxsonSz = max(mxsonSz, subsz-subtreeSz[v]);
58     if(mxsonSz < cen.w) cen = P(v, mxsonSz);
59     return cen;
60 }
61
62 void getDist(int v, int p, ll w)
63 {
64     if(w > k) return;
65     dist.push_back(w);
66     for(auto c:graph[v])
67     {
68         if(c.idx == p || isCentroid[c.idx]) continue;
69         getDist(c.idx, v, w+c.w);
70     }
71 }
72
73 ll calValidPair(int idx, ll w)
74 {
75     dist.clear();
76     getDist(idx, -1, w);
77     sort(dist.begin(), dist.end());
78     ll sum = 0;
79     for(int l = 0, r = dist.size()-1; l < r; )
80     {
81         if(dist[r]+dist[l] <= k) sum += r-l, l++;
82         else r--;
83     }
84     return sum;
85 }
86
87 ll treedc(int v)
88 {
89     ll sum = 0;
90     // find centroid
91     calSubsz(v, v);
92     int cen = getCentroid(v, v, subtreeSz[v]).idx;
93     isCentroid[cen] = true;
94
95     sum += calValidPair(cen, 0);
96     for(auto c:graph[cen])
97     {
98         if(isCentroid[c.idx]) continue;
99         sum += calValidPair(c.idx, c.w);
100         sum += treedc(c.idx);
101     }
102     return sum;
103 }
104
105 int main()
106 {
107     while(scanf("%d %lld", &vn, &k) && vn && k)
108 
```

```

109     {
110         init();
111         buildTree();
112         printf("%lld\n", treedc(1));
113     }
114     return 0;
115 }

```

2.2 nearestDist

```

1 bool cmp_y(P a, P b)
2 {
3     return a.y < b.y;
4 }
5
6 bool cmp_x(P a, P b)
7 {
8     return a.x < b.x;
9 }
10
11 double dc(P *arr, int n)
12 {
13     if(n == 1) return INF;
14     int mid = n/2;
15     double cx = arr[mid].x;
16     double dist = min( dc(arr, mid), dc(arr+mid, n-mid) );
17     inplace_merge(arr, arr+mid, arr+n, cmp_y);
18     static vector<P> brr; brr.clear();
19     for(int i = 0; i < n; i++)
20     {
21         if(fabs(arr[i].x)-cx >= dist) continue;
22         for(int j = brr.size()-1; j >= 0; j--)
23         {
24             double dx = brr[j].x-arr[i].x;
25             double dy = brr[j].y-arr[i].y;
26             if(fabs(dy) >= dist) break;
27             dist = min(dist, sqrt(dx*dx+dy*dy));
28         }
29         brr.push_back(arr[i]);
30     }
31     return dist;
32 }
33
34 double nearestDist(P *arr, int n)
35 {
36     sort(arr, arr+n, cmp_x);
37     return dc(arr, n);
38 }

```

3 Flow

3.1 dinic

```

1 template<typename T>
2 struct DINIC{
3     static const int MAXN=105;
4     static const T INF=INT_MAX;
5     int n, level[MAXN], cur[MAXN];

```

```

6 struct edge{
7     int v,pre;
8     T cap,flow,r;
9     edge(int v,int pre,T cap):v(v),pre(pre),cap(cap),flow(0),r(cap){}
10 };
11 int g[MAXN];
12 vector<edge> e;
13 void init(int _n){
14     memset(g,-1,sizeof(int)*(n+1));
15     e.clear();
16 }
17 void add_edge(int u,int v,T cap,bool directed=false){
18     e.push_back(edge(v,g[u],cap));
19     g[u]=e.size()-1;
20     e.push_back(edge(u,g[v],directed?0:cap));
21     g[v]=e.size()-1;
22 }
23 int bfs(int s,int t){
24     memset(level,0,sizeof(int)*(n+1));
25     memcpy(cur,g,sizeof(int)*(n+1));
26     queue<int> q;
27     q.push(s);
28     level[s]=1;
29     while(q.size()){
30         int u=q.front();q.pop();
31         for(int i=g[u];~i;i=e[i].pre){
32             if(!level[e[i].v]&&e[i].r){
33                 level[e[i].v]=level[u]+1;
34                 q.push(e[i].v);
35                 if(e[i].v==t) return 1;
36             }
37         }
38     }
39     return 0;
40 }
41 T dfs(int u,int t,T cur_flow=INF){
42     if(u==t) return cur_flow;
43     T df;
44     for(int &i=cur[u];~i;i=e[i].pre){
45         if(level[e[i].v]==level[u]+1&&e[i].r){
46             if(df=dfs(e[i].v,t,min(cur_flow,e[i].r))){
47                 e[i].flow+=df;
48                 e[i^1].flow-=df;
49                 e[i].r-=df;
50                 e[i^1].r+=df;
51                 return df;
52             }
53         }
54     }
55     return level[u]=0;
56 }
57 T dinic(int s,int t,bool clean=true){
58     if(clean){
59         for(size_t i=0;i<e.size();++i){
60             e[i].flow=0;
61             e[i].r=e[i].cap;
62         }
63     }
64     T ans=0, mf=0;
65     while(bfs(s,t)) while(mf=dfs(s,t)) ans+=mf;
66     return ans;
67 }
68 };

```

3.2 minCostMaxFlow

```

1 template<typename _T>
2 struct MCMF{
3     static const int MAXN=440;
4     static const _T INF=999999999;
5     struct edge{
6         int v,pre;
7         _T cap,cost;
8         edge(int v,int pre,_T cap,_T cost):v(v),pre(pre),cap(cap),cost(cost){}
9     };
10    int n,S,T;
11    _T dis[MAXN],piS,ans;
12    bool vis[MAXN];
13    vector<edge> e;
14    int g[MAXN];
15    void init(int _n){
16        memset(g,-1,sizeof(int)*(n+1));
17        e.clear();
18    }
19    void add_edge(int u,int v,_T cap,_T cost,bool directed=false){
20        e.push_back(edge(v,g[u],cap,cost));
21        g[u]=e.size()-1;
22        e.push_back(edge(u,g[v],directed?0:cap,-cost));
23        g[v]=e.size()-1;
24    }
25    _T augment(int u,_T cur_flow){
26        if(u==T||!cur_flow) return ans+=piS*cur_flow,cur_flow;
27        vis[u]=1;
28        _T r=cur_flow,d;
29        for(int i=g[u];~i;i=e[i].pre){
30            if(e[i].cap&&e[i].cost&&!vis[e[i].v]){
31                d=augment(e[i].v,min(r,e[i].cap));
32                e[i].cap-=d;
33                e[i^1].cap+=d;
34                if(!(r-=d)) break;
35            }
36        }
37        return cur_flow-r;
38    }
39    bool modlabel(){
40        for(int u=0;u<=n;++u) dis[u]=INF;
41        static deque<int> q;
42        dis[T]=0,q.push_back(T);
43        while(q.size()){
44            int u=q.front();q.pop_front();
45            _T dt;
46            for(int i=g[u];~i;i=e[i].pre){
47                if(e[i^1].cap&&(dt=dis[u]-e[i].cost)<dis[e[i].v]){
48                    if((dis[e[i].v]=dt)<=dis[q.size()?q.front():S]){
49                        q.push_front(e[i].v);
50                    } else q.push_back(e[i].v);
51                }
52            }
53        }
54        for(int u=0;u<=n;++u)
55            for(int i=g[u];~i;i=e[i].pre)
56                e[i].cost+=dis[e[i].v]-dis[u];
57        return piS+=dis[S], dis[S]<INF;
58    }
59    _T mincost(int s,int t){
60        S=s,T=t;
61        piS=ans=0;
62        while(modlabel()){
63            do memset(vis,0,sizeof(bool)*(n+1));

```

```

64     while (augment(S, INF));
65 } return ans;
66 }
67 };

```

4 Graph

4.1 floyd_warshall

```

1 int d[N][N];
2
3 void init()
4 {
5     for(int i = 0; i < v; i++)
6         for(int j = 0; j < v; j++)
7             if(i == j) d[i][j] = 0;
8             else d[i][j] = INF;
9 }
10
11 void floyd_warshall()
12 {
13     for(int k = 0; k < v; k++)
14         for(int i = 0; i < v; i++)
15             for(int j = 0; j < v; j++)
16                 if(d[i][k] != INF && d[k][j] != INF)
17                     d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
18 }

```

4.2 mst_prim

```

1 int cost[100][100];
2 bool used[100];
3 int mincost[100];
4 int v, e;
5 #define INF 2147483647
6
7 int prim()
8 {
9     for(int i = 0; i < v; i++)
10     {
11         mincost[i] = INF;
12         used[i] = false;
13     }
14     mincost[0] = 0;
15     int res = 0;
16
17     while(true)
18     {
19         int x = -1;
20         for(int u = 0; u < v; u++)
21             if(!used[u] && (x == -1 || mincost[u] < mincost[x])) x = u;
22
23         if(x == -1) break;
24         used[x] = true;
25         res += mincost[x];
26
27         for(int u = 0; u < v; u++)
28             mincost[u] = min(mincost[u], cost[x][u]);

```

```

29     }
30     return res;
31 }
32
33 void init()
34 {
35     for(int i = 0; i < v; i++)
36         for(int j = 0; j < v; j++)
37             if(i == j) cost[i][j] = 0;
38             else cost[i][j] = INF;
39 }

```

4.3 mst_kruskal

```

1 struct edge { int u, v, cost; };
2
3 bool comp(const edge& e1, const edge& e2)
4 {
5     return e1.cost < e2.cost;
6 }
7
8 int kruskal()
9 {
10     sort(es, es + e, comp);
11     dset s(v);
12     int res = 0;
13     for(int i = 0; i < e; i++)
14     {
15         edge E = es[i];
16         if(s.Find(E.u) != s.Find(E.v))
17         {
18             s.Union(E.u, E.v);
19             res += E.cost;
20         }
21     }
22     return res;
23 }

```

4.4 bellman_Ford

```

1 struct edge{ int from, to, cost; };
2 #define INF 2147483647
3
4 edge es[100];
5
6 int d[100]; //min distance
7 int V, E, s, f;
8
9 bool bellman_ford() // return true if there is negative loop
10 {
11     for(int i = 0; i < V; i++) d[i] = INF;
12     d[s] = 0;
13
14     for(int i = 0; i < V; i++)
15     {
16         for(int j = 0; j < E; j++)
17         {
18             edge e = es[j];
19             if(d[e.from] != INF && d[e.to] > d[e.from] + e.cost)
20             {

```

```

21         d[e.to] = d[e.from] + e.cost;
22         if(i == V - 1) return true; //got neg loop
23     }
24     if(d[e.to] != INF && d[e.from] > d[e.to] + e.cost)
25     {
26         d[e.from] = d[e.to] + e.cost;
27         if(i == V - 1) return true; //got neg loop
28     }
29 }
30 }
31 return false;
32 }

```

4.5 dijkstra

```

1 struct edge{int to, cost;};
2 typedef pair<int, int> P; //first = min distance, second = v id
3 #define f first
4 #define s second
5 #define INF 2147483647
6
7 int V, E, S, F;
8 vector<edge> G[100];
9 int d[100];
10
11 void dijkstra()
12 {
13     priority_queue<P, vector<P>, greater<P>> q;
14     fill(d, d + V, INF);
15     d[S] = 0;
16     q.push(P(0, S));
17
18     while(!q.empty())
19     {
20         P p = q.top(); q.pop();
21         int v = p.s;
22         if(d[v] < p.f) continue;
23         for(int i = 0; i < G[v].size(); i++)
24         {
25             edge e = G[v][i];
26             if(d[e.to] > d[v] + e.cost)
27             {
28                 d[e.to] = d[v] + e.cost;
29                 q.push(P(d[e.to], e.to));
30             }
31         }
32     }
33 }
34 }

```

4.6 spfa

```

1 typedef pair<int, ll> P;
2 #define idx first
3 #define w second
4 int vn, en;
5 vector<P> graph[N];
6 ll dist[N];
7
8 bool spfa() // return true if neg cycle

```

```

9 {
10     for(int i = 0; i < vn; i++) dist[i] = INF; dist[0] = 0;
11     int cnt[N] = {0};
12     bool inq[N] = {false};
13     queue<int> q; q.push(0); inq[0] = true;
14     while(!q.empty())
15     {
16         int s = q.front(); q.pop();
17         inq[s] = false;
18         for(auto e:graph[s])
19         {
20             if(dist[e.idx] > dist[s]+e.w)
21             {
22                 dist[e.idx] = dist[s]+e.w;
23                 if(++cnt[e.idx] >= vn) return true;
24                 if(!inq[e.idx])
25                 {
26                     inq[e.idx] = true;
27                     q.push(e.idx);
28                 }
29             }
30         }
31     }
32     return false;
33 }

```

5 Math

5.1 exgcd

```

1 ll exgcd(ll a, ll &ar, ll b) //維護 a*ar+b*as=gcd(a, b)
2 {
3     ll as = 0, br = 0, bs = 1;
4     while(a && b)
5     {
6         ar -= br*(a/b);
7         as -= bs*(a/b);
8         a %= b;
9         if(a == 0) break;
10        br -= ar*(b/a);
11        bs -= as*(b/a);
12        b %= a;
13    }
14    if(a == 0) a = b, ar = br; //維護 a*ar+b*as=gcd(a, b)
15    return a; //return gcd(a, b)
16 }

```

5.2 NTT

```

1 typedef long long ll;
2
3 const ll P = (479<<21)+1;
4 const ll G = 3;
5 inline ll fpw(ll x, ll y, ll m)
6 {
7     ll rtn = 1;
8     for(x=(x>=m?x%m:x);y>=1)

```

```

9   {
10   if(y&1) rtn = rtn*x%m;
11   x = x*x%m;
12   }
13   return rtn;
14 }
15 inline vector<ll> ntt(vector<ll> rtn, int Rev = 1)
16 {
17   int ntt_n = rtn.size();
18   for(int i=0,j=0;i<ntt_n;i++)
19   {
20     if(i>j) swap(rtn[i],rtn[j]);
21     for(int k=(ntt_n>>1); (j^=k)<k;k>>=1);
22   }
23   for(int i=2,m=1;i<=ntt_n;i<=1,m++)
24   {
25     ll w = 1, wn = fpw(G, (P-1)>>m,P), u, t;
26     int mh = i>>1;
27     for(int j=0;j<mh;j++)
28     {
29       for(int k=j;k<ntt_n;k+=i)
30       {
31         u = rtn[k], t = w*rtn[k+mh]%P;
32         rtn[k] = (u+t)%P;
33         rtn[k+mh] = (u-t+P)%P;
34       }
35       w = w*wn%P;
36     }
37   }
38   if(!~Rev)
39   {
40     for(int i=1;i<ntt_n/2;i++) swap(rtn[i],rtn[ntt_n-i]);
41     ll Revn = fpw(ntt_n,P-2,P);
42     for(int i=0;i<ntt_n;i++) rtn[i] = rtn[i]*Revn%P;
43   }
44   return rtn;
45 }

```

5.3 BigInteger

```

1 class BigInt
2 {
3 public:
4   // constructors
5   BigInt();
6   BigInt(ll);
7   BigInt(const char*);
8   BigInt(string);
9   BigInt(bool, vector<int>);
10  // functions
11  inline void print();
12  // operators
13  bool operator== (const BigInt &a) const;
14  bool operator!= (const BigInt &a) const;
15  bool operator< (const BigInt &a) const;
16  bool operator> (const BigInt &a) const;
17  bool operator<= (const BigInt &a) const;
18  bool operator>= (const BigInt &a) const;
19  BigInt operator- () const;
20  BigInt operator+ (const BigInt &a) const;
21  BigInt operator- (const BigInt &a) const;
22  BigInt operator* (const BigInt &a) const;
23  BigInt operator/ (const BigInt &a) const;

```

```

24   BigInt operator% (const BigInt &a) const;
25   // variables
26   const static int MAX = 1000000;
27   bool Neg = false;
28   vector<int> seq;
29 };
30
31 // constructors
32 BigInt::BigInt() {}
33 BigInt::BigInt(ll in)
34 {
35   if(in<0) Neg=true, in=-in;
36   while(in!=0)
37   {
38     seq.emplace_back(in%MAX);
39     in/=MAX;
40   }
41   if(seq.empty()) seq.emplace_back(0);
42 }
43 BigInt::BigInt(const char *s)
44 {
45   int i, j, tmp, end=0;
46   if(s[0]=='-') Neg=true, end=1;
47   for(i=strlen(s)-1, j=1, tmp=0; i>=end; i--, j*=10)
48   {
49     if(j==MAX)
50     {
51       seq.emplace_back(tmp);
52       j=1, tmp=0;
53     }
54     tmp += (s[i]-'0')*j;
55   } seq.emplace_back(tmp);
56 }
57 BigInt::BigInt(string s):BigInt(s.c_str()) {}
58 BigInt::BigInt(bool b, vector<int> v):Neg(b),seq(v) {}
59 // functions
60 void BigInt::print()
61 {
62   if(Neg) putchar('-');
63   printf("%d",seq.back());
64   for(int i=(int)(seq.size()-2); i>=0; i--)
65     printf("%06d",seq[i]);
66   puts("");
67 }
68 // operators
69 bool BigInt::operator== (const BigInt &a) const
70 {
71   return Neg==a.Neg&&seq==a.seq;
72 }
73 bool BigInt::operator!= (const BigInt &a) const
74 {
75   return !((~this)==a);
76 }
77 bool BigInt::operator< (const BigInt &a) const
78 {
79   if(Neg^a.Neg) return Neg;
80   if(seq.size()!=a.seq.size()) return Neg^(seq.size()<a.seq.size());
81   for(int i=seq.size()-1; i>=0; i--)
82     if(seq[i]!=a.seq[i]) return Neg^(seq[i]<a.seq[i]);
83   return false;
84 }
85 bool BigInt::operator> (const BigInt &a) const
86 {
87   if(Neg^a.Neg) return a.Neg;
88   if(seq.size()!=a.seq.size()) return a.Neg^(seq.size()>a.seq.size());
89   for(int i=seq.size()-1; i>=0; i--)

```

```

90     if(seq[i]!=a.seq[i]) return a.Neg^(seq[i]>a.seq[i]);
91     return false;
92 }
93 bool BigInt::operator<= (const BigInt &a) const
94 {
95     return !((*this)>a);
96 }
97 bool BigInt::operator>= (const BigInt &a) const
98 {
99     return !((*this)<a);
100 }
101 BigInt BigInt::operator- () const
102 {
103     return BigInt(Neg^1, seq);
104 }
105 BigInt BigInt::operator+ (const BigInt &a) const
106 {
107     if (Neg^a.Neg)
108         return Neg?a-(-(*this)):(*this)-(-a);
109     BigInt rtn(Neg, vector<int>(max(seq.size(),a.seq.size())));
110     for(int i=0; i<(int)(seq.size()); i++) rtn.seq[i]+=seq[i];
111     for(int i=0; i<(int)(a.seq.size()); i++) rtn.seq[i]+=a.seq[i];
112     for(int i=0; i<(int)(rtn.seq.size()-1; i++)
113         if(rtn.seq[i]>=MAX)
114             rtn.seq[i+1]+=rtn.seq[i]/MAX, rtn.seq[i]%=MAX;
115     if(rtn.seq.back()>=MAX)
116     {
117         rtn.seq.emplace_back(rtn.seq.back()/MAX);
118         rtn.seq[rtn.seq.size()-2]%=MAX;
119     }
120     return rtn;
121 }
122 BigInt BigInt::operator- (const BigInt &a) const
123 {
124     if (Neg^a.Neg) return (*this)+(-a);
125     if (Neg^((*this)<a)) return (-a)-(-(*this));
126     BigInt rtn(Neg, vector<int>(max(seq.size(),a.seq.size())));
127     for(int i=0; i<(int)(seq.size()); i++) rtn.seq[i]+=seq[i];
128     for(int i=0; i<(int)(a.seq.size()); i++) rtn.seq[i]-=a.seq[i];
129     for(int i=0; i<(int)(rtn.seq.size()-1; i++)
130         if(rtn.seq[i]<0)
131             rtn.seq[i+1]--, rtn.seq[i]+=MAX;
132     while(!rtn.seq.empty()&&!rtn.seq.back()) rtn.seq.pop_back();
133     if(rtn.seq.empty()) rtn = BigInt(011);
134     return rtn;
135 }
136 BigInt BigInt::operator* (const BigInt &a) const
137 {
138     BigInt rtn(Neg^a.Neg, vector<int>(0));
139     vector<Complex> x, y;
140     for(auto i:seq) x.emplace_back(i);
141     for(auto i:a.seq) y.emplace_back(i);
142     int N=1;
143     while(N<(int)(x.size()+y.size())) N <= 1;
144     while(N!=(int)(x.size())) x.emplace_back(0);
145     while(N!=(int)(y.size())) y.emplace_back(0);
146     x = fft(x), y = fft(y);
147     for(int i=0; i<N; i++) x[i] = x[i]*y[i];
148     x = fft(x,-1);
149     ll tmp = 0;
150     for(int i=0; i<N; i++)
151     {
152         tmp += (ll)(x[i].x+0.1);
153         rtn.seq.emplace_back(tmp%MAX);
154         tmp /= MAX;
155     } rtn.seq.emplace_back(tmp);

```

```

156     while(!rtn.seq.empty()&&!rtn.seq.back()) rtn.seq.pop_back();
157     if(rtn.seq.empty()) rtn = BigInt(011);
158     return rtn;
159 }
160 BigInt BigInt::operator/ (const BigInt &a) const
161 {
162     if(a==BigInt(011)) return a;
163     BigInt rtn, check, Btmp, posiA, posiB;
164     posiA = (*this), posiA.Neg = false;
165     posiB = a, posiB.Neg = false;
166     int PRECISION = max(seq.size(),a.seq.size()+6, N = 1;
167     ll tmp = 0;
168     while(N<PRECISION+6) N <= 1;
169     N <= 1;
170     vector<Complex> B, c1(N,0), c2(N,0), c3(N,0);
171     vector<Complex> *x = (&c1), *xp = (&c2), *calc = (&c3);
172     for(int i=a.seq.size()-1; i>=0; i--)
173         B.emplace_back(a.seq[i]);
174     B.resize(N,0);
175     B = fft(B);
176     (*x)[a.seq.size()] = MAX/a.seq.back();
177     bool found = false;
178     while(!found)
179     {
180         (*x) = fft(*x);
181         for(int i=0; i<N; i++) (*calc)[i] = (*x)[i]*B[i];
182         (*calc) = fft(*calc,-1);
183         for(int i=a.seq.size()-1; i<N; i++)
184             (*calc)[i-a.seq.size()+1] = (*calc)[i];
185         for(int i=N-a.seq.size()+1; i<N; i++)
186             (*calc)[i] = 0;
187         for(int i=N-1; i>=1; i--)
188         {
189             tmp = (ll)((*calc)[i].x+0.1);
190             (*calc)[i-1] = (ll)((*calc)[i-1].x+0.1)+tmp/MAX;
191             (*calc)[i] = tmp % MAX;
192             if(tmp)
193             {
194                 (*calc)[i-1] = (ll)((*calc)[i-1].x+0.1)+1;
195                 (*calc)[i] = MAX-tmp;
196             }
197         } (*calc)[0] = 2-(ll)((*calc)[0].x+0.1);
198         for(int i=PRECISION+6; i<N; i++) (*calc)[i] = 0;
199         (*calc) = fft(*calc);
200         for(int i=0; i<N; i++) (*xp)[i] = (*calc)[i]*(*x)[i];
201         (*xp) = fft(*xp,-1);
202         (*x) = fft(*x,-1);
203         for(int i=N-1; i>=1; i--)
204         {
205             tmp = (ll)((*xp)[i].x+0.1);
206             (*xp)[i] = tmp%MAX;
207             (*xp)[i-1] = (ll)((*xp)[i-1].x+0.1)+tmp/MAX;
208         }
209         for(int i=PRECISION+6; i<N; i++) (*xp)[i] = 0;
210         found = true;
211         for(int i=0; i<=PRECISION&&found; i++)
212             if((ll)((*xp)[i].x+0.1) != (ll)((*x)[i].x+0.1))
213                 found = false;
214         calc = x, x = xp, xp = calc, calc = (&c3);
215     }
216     for(int i=N-1; i>=(int)(a.seq.size()-1; i--)
217         rtn.seq.emplace_back((ll)((*x)[i].x+0.1));
218     while(!rtn.seq.back()) rtn.seq.pop_back();
219     rtn = rtn*posiA;
220     for(int i=N-1; i<(int)(rtn.seq.size()); i++)
221         rtn.seq[i-N+1] = rtn.seq[i];

```

```

222 for(int i=max((int)(rtn.seq.size()-N+1),0); i<(int)(rtn.seq.size()); i++)
223     rtn.seq[i] = 0;
224 while(!rtn.seq.empty() && !rtn.seq.back()) rtn.seq.pop_back();
225 if(rtn.seq.empty()) rtn = BigInt(0);
226 check = rtn*posiB;
227 BITmp = check+posiB;
228 while(posiA>=BITmp)
229     rtn = rtn+BigInt(1), check = BITmp, BITmp = check+posiB;
230 BITmp = check-posiB;
231 while(posiA<=BITmp)
232     rtn = rtn-BigInt(1), check = BITmp, BITmp = check-posiB;
233 rtn.Neg = Neg^a.Neg;
234 return rtn;
235 }
236 BigInt BigInt::operator% (const BigInt &a) const
237 {
238     return (*this) - ((*this)/a) *a;
239 }

```

5.4 prime_detect

```

1 const int N = 10000000;
2 bool isprime[N] = {true};
3 void prime_detect()
4 {
5     for(int i = 2; i < sq; i++)
6         if(isprime[i])
7             for(int j = i*i; j < N; j+=i) isprime[j] = false;
8 }

```

5.5 modeq

```

1 // 解線性模方程組 (最小非負整數解)
2 const int N; // N個方程
3 ll A[N], B[N], M[N]; // A * X = B (%M)
4 ll solve() // 解X, return INF if no solution
5 {
6     ll k = 0, h = 1;
7     for(ll i = 0; i < N; i++)
8     {
9         ll a = A[i]*h, b = B[i]-A[i]*k, m = M[i], ar;
10        ll d = exgcd(a, ar=1, m);
11        if(b%d != 0) return INF;
12        ll n = abs(m/d);
13        ll t = ar*b/d; t%=n; t+=n; t%=n;
14
15        k += h*t, h *= n; k%=h; // 維護解是正的
16    }
17    int ret = (k%h+h)%h;
18    return ret;
19 }

```

5.6 FFT

```

1 const double PI = acos(-1.0);
2 struct Complex
3 {
4     double x,y;
5     Complex() {}
6     Complex(double a):x(a),y(0) {}
7     Complex(double a, double b):x(a),y(b) {}
8     Complex operator+ (const Complex &a) { return Complex(x+a.x,y+a.y); }
9     Complex operator- (const Complex &a) { return Complex(x-a.x,y-a.y); }
10    Complex operator* (const Complex &a) { return Complex(x*a.x-y*a.y,x*a.y+y*a.x); }
11 };
12 inline vector<Complex> fft(vector<Complex> rtn, int Rev = 1)
13 {
14     int fft_n = rtn.size();
15     for(int i=0,j=0;i<fft_n;i++)
16     {
17         if(i>j) swap(rtn[i],rtn[j]);
18         for(int k=(fft_n>>1); (j^=k)<k;k>=1);
19     }
20     for(int i=2,m;i<=fft_n;i<=1)
21     {
22         m = i>>1;
23         for(int j=0;j<fft_n;j+=i)
24         {
25             for(int k=0;k<m;k++)
26             {
27                 Complex y = rtn[j+k+m]*Complex(cos(2*PI/i*k), Rev*sin(2*PI/i*k));
28                 rtn[j+k+m] = rtn[j+k]-y;
29                 rtn[j+k] = rtn[j+k]+y;
30             }
31         }
32     }
33     for(int i=0;i<~Rev&&i<fft_n;i++)
34         rtn[i].x = rtn[i].x/fft_n;
35     return rtn;
36 }

```

6 String

6.1 SuffixArray-STL

```

1 struct CMP
2 {
3     int len,k,*Rank,a,b;
4     inline bool operator()(int i, int j)
5     {
6         if(Rank[i]!=Rank[j]) return Rank[i]<Rank[j];
7         a=(i+k)<len?Rank[i]:-1;
8         b=(j+k)<len?Rank[j]:-1;
9         return a<b;
10    }
11 };
12 void SA_build(int *SA, int *Rank, char *S) {
13     int tmp[MAX_N], len=strlen(S);
14     for(int i=0;i<len;i++) SA[i]=i, Rank[i]=S[i];
15     CMP cmp={len,1};
16     while(cmp.k*=2)
17     {
18         cmp.Rank=Rank;
19         sort(SA,SA+len,cmp);
20         tmp[SA[0]]=0;

```



```

21     for(int i=1;i<len;i++)
22         tmp[SA[i]]=tmp[SA[i-1]]+cmp(SA[i-1],SA[i]);
23     if(tmp[SA[len-1]]==len-1) break;
24     for(int i=0;i<len;i++) Rank[i]=tmp[i];
25 }
26 }

```

6.2 Z-value

```

1 void Z_build(const char *S, int *Z)
2 {
3     Z[0]=0;
4     int b=0;
5     for(int i=1;S[i];i++)
6     {
7         if(Z[b]+b<i) Z[i]=0;
8         else Z[i]=min(Z[b]+b-i,Z[i-b]);
9         while(S[i+Z[i]]&&S[Z[i]]==S[i+Z[i]]) Z[i]++;
10        if(Z[i]+i>Z[b]+b) b=i;
11    }
12 }

```

6.3 LCP

```

1 //build query in O(nlogn), query LCP(i,j) in O(1)
2 int dp_height[MAX_N][20];
3 void height_build(int *SA, int *Rank, char *S, int *Height)
4 {
5     int len=strlen(S), k=0;
6     for(int i=0;i<len;i++)
7     {
8         if(Rank[i]==0) continue;
9         while(S[i+k]==S[SA[Rank[i]-1]+k]) k++;
10        Height[Rank[i]]=k;
11        if(k) k--;
12    } Height[0]=0;
13    for(int i=0;i<len;i++) dp_height[i][0]=Height[i];
14    for(int i=0;i<len;i++) for(int j=1;i+(1<<j)<len;j++)
15        dp_height[i][j]=min(dp_height[i][j-1], dp_height[i+(1<<(j-1))][j-1]);
16 }
17 int height_query(int x, int y)
18 {
19     int k=0;
20     while((1<<(k+1))<=y-x) k++;
21     return min(dp_height[x+1][k], dp_height[y-(1<<k)+1][k]);
22 }

```

6.4 SuffixArray

```

1 void SA_radix_sort(int *s, int *e, int *Rank, int rankcnt)
2 {
3     int box[MAX_N], tmp[MAX_N], len=e-s;
4     memset(box,0,sizeof(int)*rankcnt);
5     for(int i=0;i<len;i++) box[Rank[i]]++;
6     for(int i=1;i<rankcnt;i++) box[i]=box[i]+box[i-1];
7     for(int i=len-1;i>=0;i--) tmp[--box[Rank[s[i]]]]=s[i];

```

```

8     for(int i=0;i<len;i++) s[i]=tmp[i];
9 }
10 #define equal(a,b,c) c[a]!=c[b]||a+k>=len||c[a+k]!=c[b+k]
11 void SA_build(int *SA, int *Rank, char *S)
12 {
13     int ranktmp[MAX_N], len=strlen(S), rankcnt='z'+1;
14     for(int i=0;i<len;i++) Rank[i]=S[i];
15     for(int k=1;rankcnt!=len;k*=2)
16     {
17         for(int i=0;i<len;i++) SA[i]=(i+len-k)%len;
18         SA_radix_sort(SA+k, SA+len, Rank+k, rankcnt);
19         SA_radix_sort(SA, SA+len, Rank, rankcnt);
20         ranktmp[SA[0]]=0, rankcnt=0;
21         for(int i=1;i<len;i++)
22             ranktmp[SA[i]]=rankcnt+equal(SA[i-1], SA[i], Rank);
23         rankcnt++;
24         for(int i=0;i<len;i++) Rank[i]=ranktmp[i];
25     }
26 }
27 #undef equal

```

6.5 KMP

```

1 void failure_build(const char *p, int *fail)
2 {
3     for(int i=1, j=fail[0]=-1; p[i]; i++)
4     {
5         while(j>=0&&p[j+1]!=p[i]) j=fail[j];
6         if(p[j+1]==p[i]) j++;
7         fail[i]=j;
8     }
9 }
10 int KMP(const char *T, const char *P, int *fail)
11 {
12     failure_build(P, fail);
13     for(int i=0, j=-1; T[i]; i++)
14     {
15         while(j>=0&&P[j+1]!=T[i]) j=fail[j];
16         if(P[j+1]==T[i]) j++;
17         if(!P[j+1]) return i-j;
18     }
19     return -1;
20 }
21
22 //使用方法: KMP(主字串, 待匹配字串, failure array)
23 //回傳: 第一個完全匹配的位置

```

7 Tree

7.1 treap

```

1 struct Treap
2 {
3     int pri, sz;
4     int rev;
5     ll data, sum;    // tag: make-same
6     Treap *lchild, *rchild;

```

```

7   Treap(l1 d):pri(rand()), sz(1), rev(0), data(d), sum(d), lchild(NULL), rchild(NULL)
8   {
9   }
10  inline void up();
11  inline void down();
12  };
13
14  inline int size(Treap *t) { return t? t->sz:0; }
15  inline l1 get_data(Treap *t) { return t? t->data:0; }
16  inline l1 get_sum(Treap *t) { return t? t->sum:0; }
17
18  inline void Treap::up()
19  {
20      if(lchild) lchild->down();
21      if(rchild) rchild->down();
22      sz = 1+size(lchild)+size(rchild);
23      sum = get_sum(lchild) + data + get_sum(rchild);
24  }
25
26  inline void Treap::down()
27  {
28      if(rev)
29      {
30          swap(mxpre, mxpost);
31          swap(lchild, rchild);
32          if(lchild) lchild->rev ^= 1;
33          if(rchild) rchild->rev ^= 1;
34          rev ^= 1;
35      }
36  }
37
38  Treap *merge(Treap *a, Treap *b)
39  {
40      if(!a || !b) return (a? a:b);
41      if(a->pri < b->pri)
42      {
43          a->down();
44          a->rchild = merge(a->rchild, b);
45          a->up();
46          return a;
47      }
48      else
49      {
50          b->down();
51          b->lchild = merge(a, b->lchild);
52          b->up();
53          return b;
54      }
55  }
56
57  void split(Treap *o, Treap *&a, Treap *&b, int k)
58  {
59      if(!o) a = b = NULL;
60      else
61      {
62          o->down();
63          if(k >= size(o->lchild)+1)
64          {
65              a = o;
66              split(o->rchild, a->rchild, b, k-size(o->lchild)-1);
67          }
68          else
69          {
70              b = o;
71              split(o->lchild, a, b->lchild, k);
72          }

```

```

73      o->up();
74  }
75  }

```

7.2 HeavyLightDecomposition

```

1  const int MAX_N;
2  vector<int> link[MAX_N]; //edge
3
4  void dfs_build(int now, int fa, int *weight, int *depth, int *pa, int *son)
5  {
6      weight[now]=1;
7      son[now]=-1;
8      pa[now]=fa;
9      for(auto i:link[now])
10     {
11         if(i==fa) continue;
12         depth[i]=depth[now]+1;
13         dfs_build(i,now,weight,depth,pa,son);
14         if(son[now]==-1 || weight[son[now]]<weight[i]) son[now]=i;
15         weight[now]+=weight[i];
16     }
17 }
18 void build_top(int now, int top,int *pa, int *son, int *link_top)
19 {
20     link_top[now]=top;
21     if(son[now]==-1) return;
22     build_top(son[now],top,pa,son,link_top);
23     for(auto i:link[now])
24     {
25         if(i==son[now] || i==pa[now]) continue;
26         build_top(i,i,pa,son,link_top);
27     }
28 }
29 inline void HLD(int *weight, int *depth, int *pa, int *son, int *link_top)
30 {
31     memset(son,-1,sizeof(int)*MAX_N);
32     depth[1]=1; //set node(1) as root
33     dfs_build(1,0,weight,depth,pa,son);
34     build_top(1,1,pa,son,link_top);
35 }
36 inline int find_lca(int x, int y, int *depth, int *pa, int *link_top)
37 {
38     int tx=link_top[x], ty=link_top[y];
39     while(tx!=ty)
40     {
41         if(depth[tx]<depth[ty])
42         {
43             swap(tx,ty);
44             swap(x,y);
45         }
46         tx=link_top[x=pa[x]];
47     }
48     return depth[x]<depth[y]?x:y;
49 }
50 //usage:
51 //build HeavyLightDecomposition: HLD
52 //find LCA(x,y): find_lca

```

7.3 disjoint_set

```

1 // path compression
2 int f[N];
3
4 int findrt(int x)
5 {
6     if(f[x] == x) return x;
7     else return f[x] = findrt(f[x]);
8 }
9
10 int same(int x, int y)
11 {
12     return findrt(x) == findrt(y);
13 }
14
15 void uni(int x, int y)
16 {
17     f[findrt(y)] = findrt(x);
18 }
19
20 void init()
21 {
22     for(int i = 0; i < N; i++) f[i] = i;
23 }
24
25 //union by rank
26 int f[N]; //disjoint set
27 int rk[N]; //union by rank
28
29 int findrt(int x)
30 {
31     if(f[x] == x) return x;
32     else return f[x] = findrt(f[x]);
33 }
34
35 bool same(int x, int y)
36 {
37     return findrt(x) == findrt(y);
38 }
39
40 void uni(int x, int y)
41 {
42     x = findrt(x), y = findrt(y);
43     if(x == y) return;
44     if(rk[x] < rk[y]) f[x] = y;
45     else if(rk[x] == rk[y]) f[x] = y, rk[y]++;
46     else f[y] = x;
47 }
48
49 void init()
50 {
51     for(int i = 0; i < N; i++) f[i] = i, rk[i] = 0;
52 }

```

7.4 2d_st_tag

```

1 //二維陣列單點查詢區間加值
2 class St1d
3 {
4 private:
5     ll st[4*N];
6
7 public:
8     void build();

```

```

9     void modify(int l, int r, int idx, int L, int R, ll v);
10    ll query(int l, int r, int idx, int x);
11    void down(int idx);
12 };
13
14 void St1d::build()
15 {
16     memset(st, 0, sizeof(st));
17 }
18
19 void St1d::modify(int l, int r, int idx, int L, int R, ll v)
20 {
21     if(r < L || R < l) return;
22     if(L <= l && r <= R)
23     {
24         st[idx] += v;
25         return;
26     }
27     assert(l != r);
28     down(idx);
29     int mid = (l+r)/2;
30     modify(l, mid, idx*2, L, R, v);
31     modify(mid+1, r, idx*2+1, L, R, v);
32 }
33
34 ll St1d::query(int l, int r, int idx, int x)
35 {
36     if(x < l || r < x) return 0;
37     if(l == x && r == x) return st[idx];
38     down(idx);
39     int mid = (l+r)/2;
40     ll left = query(l, mid, idx*2, x);
41     ll right = query(mid+1, r, idx*2+1, x);
42     return left+right;
43 }
44
45 void St1d::down(int idx)
46 {
47     st[idx*2] += st[idx], st[idx*2+1] += st[idx];
48     st[idx] = 0;
49 }
50
51 ///////////////////////////////////////////////////
52
53 class St2d
54 {
55 private:
56     St1d st[4*N];
57
58 public:
59     void build(int il, int ir, int idx);
60     void modify(int il, int ir, int jl, int jr, int idx, int iL, int iR, int jL, int jR, ll v);
61     ll query(int il, int ir, int jl, int jr, int idx, int i, int j);
62 };
63
64 void St2d::build(int il, int ir, int idx)
65 {
66     st[idx].build();
67     if(il == ir) return;
68     int mid = (il+ir)/2;
69     build(il, mid, idx*2);
70     build(mid+1, ir, idx*2+1);
71 }
72

```

```

73 void St2d::modify(int il, int ir, int jl, int jr, int idx, int iL, int iR, int jL, int jR, ll
    v)
74 {
75     if(ir < iL || iR < iL) return;
76     if(iL <= iL && iR <= iR)
77     {
78         st[idx].modify(jl, jr, 1, jL, jR, v); return;
79     }
80     int mid = (il+ir)/2;
81     modify(il, mid, jl, jr, idx*2, iL, iR, jL, jR, v);
82     modify(mid+1, ir, jl, jr, idx*2+1, iL, iR, jL, jR, v);
83 }
84
85 ll St2d::query(int il, int ir, int jl, int jr, int idx, int i, int j)
86 {
87     ll tot = 0;
88     if(i < il || ir < i) return 0;
89     if(il <= i && i <= ir) tot += st[idx].query(jl, jr, 1, j);
90     if(il == i && ir == i) return tot;
91     int mid = (il+ir)/2;
92     tot += query(il, mid, jl, jr, idx*2, i, j);
93     tot += query(mid+1, ir, jl, jr, idx*2+1, i, j);
94     return tot;
95 }

```

7.5 BIT

```

1  #define lowbit(x) x&-x
2
3  int arr[N]; //紀錄前綴和
4  int bit[N];
5
6  void conv(int a[], int n) //離散化
7  {
8      vector<int> tmp;
9      for(int i = 1; i <= n; i++) tmp.push_back(a[i]);
10     sort(tmp.begin(), tmp.end());
11     for(int i = 1; i <= n; i++) a[i] = lower_bound(tmp.begin(), tmp.end(), a[i]) - tmp.begin
        () + 1;
12 }
13
14 void buildbit() //每個bit[x]紀錄[x-lowbit(x)+1, x]的總和
15 {
16     for(int i = 0; i < n; i++) bit[i] = arr[i]-arr[i-lowbit(i)];
17 }
18
19 int sum(int x) //查詢[1,x]的總和
20 {
21     int rtn = 0;
22     for(;x>=lowbit(x); rtn += bit[x];
23     return rtn;
24 }
25
26 void modify(int x, int d) //把位置x的東西加上d
27 {
28     for(;x<=n;x+=lowbit(x)) bit[x] += d;
29 }

```

7.6 1d_segTree_tag

```

1  //線段樹懶人標記：一維陣列區間加值區間乘值區間查詢總和
2  struct Node //data = data*mul+add;
3  {
4      ll data, mul, add;
5  };
6
7  ll getval(int l, int r, int idx)
8  {
9      return (st[idx].data*st[idx].mul%MD+(r-l+1)*st[idx].add%MD)%MD;
10 }
11
12 void up(int l, int r, int idx)
13 {
14     int mid = 1+(r-l)/2;
15     st[idx].data = (getval(l, mid, idx*2)+getval(mid+1, r, idx*2+1))%MD;
16 }
17
18 void down(int l, int r, int idx)
19 {
20     st[idx].data = getval(l, r, idx);
21     int lson = idx*2, rson = idx*2+1;
22     if(l != r)
23     {
24         st[lson].mul = st[lson].mul*st[idx].mul%MD;
25         st[lson].add = (st[lson].add*st[idx].mul+st[idx].add)%MD;
26         st[rson].mul = st[rson].mul*st[idx].mul%MD;
27         st[rson].add = (st[rson].add*st[idx].mul+st[idx].add)%MD;
28     }
29     st[idx].mul = 1, st[idx].add = 0;
30 }
31
32 void buildst(int l, int r, int idx)
33 {
34     st[idx].mul = 1, st[idx].add = 0;
35     if(l == r)
36     {
37         st[idx].data = arr[l];
38         return;
39     }
40     int mid = 1+(r-l)/2;
41     buildst(l, mid, idx*2);
42     buildst(mid+1, r, idx*2+1);
43     up(l, r, idx);
44 }
45
46 void add(int l, int r, int idx, int L, int R, int v) //操作L,R
47 {
48     if(r < L || R < l) return;
49     if(L <= l && r <= R)
50     {
51         st[idx].add = (st[idx].add+v)%MD;
52         return;
53     }
54     down(l, r, idx);
55     int mid = 1+(r-l)/2;
56     add(l, mid, idx*2, L, R, v);
57     add(mid+1, r, idx*2+1, L, R, v);
58     up(l, r, idx);
59 }
60
61 void mul(int l, int r, int idx, int L, int R, int v)
62 {
63     if(r < L || R < l) return;
64     if(L <= l && r <= R)
65     {

```

```

66     st[idx].add = st[idx].add*v%MD;
67     st[idx].mul = st[idx].mul*v%MD;
68     return;
69 }
70 down(l, r, idx);
71 int mid = l+(r-l)/2;
72 mul(l, mid, idx*2, L, R, v);
73 mul(mid+1, r, idx*2+1, L, R, v);
74 up(l, r, idx);
75 }
76
77 ll query(int l, int r, int idx, int L, int R)
78 {
79     if(r < L || R < l) return 0;
80     if(L <= l && r <= R)
81     {
82         return getval(l, r, idx);
83     }
84     down(l, r, idx);
85     int mid = l+(r-l)/2;
86     return (query(l, mid, idx*2, L, R)+query(mid+1, r, idx*2+1, L, R))%MD;
87 }

```

7.7 1d_segTree

```

1 void buildst(int l, int r, int idx) //l, r是st的區間
2 {
3     if(l == r)
4     {
5         st[idx] = arr[l];
6         return;
7     }
8     int mid = (l+r)/2;
9     buildst(l, mid, idx*2);
10    buildst(mid+1, r, idx*2+1);
11    st[idx] = max(st[idx*2], st[idx*2+1]);
12 }
13
14 ll query(int l, int r, int idx, int L, int R) //L,R是操作的區間
15 {
16     if(r < L || R < l) return -INF;
17     if(L <= l && r <= R) return st[idx];
18     int mid = (l+r)/2;
19     return max(query(l, mid, idx*2, L, R), query(mid+1, r, idx*2+1, L, R));
20 }
21
22 void modify(int l, int r, int idx, int x, int v)
23 {
24     if(r < x || x < l) return;
25     if(l == r)
26     {
27         st[idx] += v; return;
28     }
29     int mid = (l+r)/2;
30     modify(l, mid, idx*2, x, v);
31     modify(mid+1, r, idx*2+1, x, v);
32     st[idx] = max(st[idx*2], st[idx*2+1]);
33 }

```

ACM ICPC
TEAM
REFERENCE - 燈
泡邪教

Contents	
1 ComputationalGeometry	1
1.1 operators	1
1.2 inter_par	1
2 DivideConquer	1
2.1 PojTree	1
2.2 nearestDist	2
3 Flow	2
3.1 dinic	2
3.2 minCostMaxFlow	3
4 Graph	4
4.1 floyd_warshall	4
4.2 mst_prim	4
4.3 mst_kruskal	4
4.4 bellman_Ford	4
4.5 dijkstra	5
4.6 spfa	5
5 Math	5
5.1 exgcd	5
5.2 NTT	5
5.3 BigInteger	6
5.4 prime_detect	8
5.5 modeq	8
5.6 FFT	8
6 String	8
6.1 SuffixArray-STL	8
6.2 Z-value	9
6.3 LCP	9
6.4 SuffixArray	9
6.5 KMP	9
7 Tree	9
7.1 treap	9
7.2 HeavyLightDecomposition . .	10
7.3 disjoint_set	10
7.4 2d_st_tag	11
7.5 BIT	12
7.6 1d_segTree_tag	12
7.7 1d_segTree	13