lightbulb CodeBook

Math

prime detection

```
const int N = 10000000;
bool isprime[N] = {true};
void prime_detect()
{
    for(int i = 2; i < sq; i++)
        if(isprime[i])
            for(int j = i*i; j < N; j+=i) isprime[j] = false;
}
```

exgcd

```
ll exgcd(ll a, ll &ar, ll b)  //維護a*ar+b*as=gcd(a, b)
{
   ll as = 0, br = 0, bs = 1;
   while(a && b)
   {
      ar -= br*(a/b);
      as -= bs*(a/b);
      a %= b;
      if(a == 0) break;
      br -= ar*(b/a);
      bs -= as*(b/a);
      b %= a;
   }
   if(a == 0) a = b, ar = br;  //維護a*ar+b*as=gcd(a, b)
   return a; //return gcd(a, b)
}
```

線性模方程組

```
const int N;  //N個方程
ll A[N], B[N], M[N];  // A * X = B (%M)
ll solve() //解X, return INF if no solution
{
   ll k = 0, h = 1;
   for(ll i = 0; i < N; i++)
   {
      ll a = A[i]*h, b = B[i]-A[i]*k, m = M[i], ar;
      ll d = exgcd(a, ar=1, m);

      if(b%d != 0) return INF;
```

```
        ll n = abs(m/d);
        ll t = ar*b/d; t%=n; t+=n; t%=n;

        k += h*t, h *= n; k%=h; //維護解是正的
    }
    int ret = (k%h+h)%h;
    return ret;
}
```

Computational Geometry

operators

```
typedef std::pair<double,double> Pt;
#define X first
#define Y second
const double eps = 1e-6;
Pt point( double x , double y ){
    return make_pair( x , y );
}
Pt operator+( const Pt& p1 , const Pt& p2 ){
  return Pt( p1.X + p2.X , p1.Y + p2.Y );
}
Pt operator-( const Pt& p1 , const Pt& p2 ){
  return Pt( p1.X - p2.X , p1.Y - p2.Y );
}
double operator*( const Pt& p1 , const Pt& p2 ){
  return p1.X * p2.X + p1.Y * p2.Y;
}
double operator^( const Pt& p1 , const Pt& p2 ){
  return p1.X * p2.Y - p1.Y * p2.X;
}
Pt operator*( const Pt& p1 , const double& k ){
  return Pt( p1.X * k , p1.Y * k );
}
Pt operator/( const Pt& p1 , const double& k ){
  return Pt( p1.X / k , p1.Y / k );
}
bool equal( const double& a , const double& b ){
  return b - eps < a && a < b + eps;
}
bool less( const double& a , const double& b ){
  return a < b - eps;
}
bool lessOrEqual( const double& a , const double& b ){
  return a < b + eps;
}
double abs( const Pt& p1 ){
  return sqrt( p1 * p1 );
}
double area(){
    double sum = 0;
```

```
    for(int i = 0; i < n; i++) sum += 0.5*p[i]^[i+1];
    return sum;
}
Pt o;
D angle( const Pt& x ){
  return atan2( x.Y , x.X );
}
bool cmp_angle( Pt a , Pt b ){
  return angle( a - o ) < angle( b - o );
}
bool cmp_cross( Pt a , Pt b ){
  return ( a - o ) ^ ( b - o ) > 0;
}
```

相交、平行、共線

```
int ori( const Pt& o , const Pt& a , const Pt& b ){
  double cross = ( a - o ) ^ ( b - o );
  if( fabs( cross ) < eps ) return 0;
  return cross > 0 ? 1 : -1;
}
bool intersect(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){ //線段p1p2, p3p4
    return ori(p1, p2, p3)^ori(p1, p2, p4) < 0;
}
int parallel(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){
    return (p2-p1)^(p4-p3) == 0;
}
bool Collinear(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){ //共線
    ori(p1, p2, p3) == 0;
}
```

Tree

disjoint set

```
// path compression
int f[N];

int findrt(int x)
{
    if(f[x] == x) return x;
    else return f[x] = findrt(f[x]);
}

int same(int x, int y)
{
    return findrt(x) == findrt(y);
}

```

```
void uni(int x, int y)
{
    f[findrt(y)] = findrt(x);
}

void init()
{
    for(int i = 0; i < N; i++) f[i] = i;
}

//union by rank
int f[N];  //disjoint set
int rk[N]; //union by rank

int findrt(int x)
{
    if(f[x] == x) return x;
    else return f[x] = findrt(f[x]);
}

bool same(int x, int y)
{
    return findrt(x) == findrt(y);
}

void uni(int x, int y)
{
    x = findrt(x), y = findrt(y);
    if(x == y) return;
    if(rk[x] < rk[y]) f[x] = y;
    else if(rk[x] == rk[y]) f[x] = y, rk[y]++;
    else f[y] = x;
}

void init()
{
    for(int i = 0; i < N; i++) f[i] = i, rank[i] = 0;
}
```

1d segment tree

```
void buildst(int l, int r, int idx)  //l, r是st的區間
{
    if(l == r)
    {
        st[idx] = arr[l];
        return;
    }
    int mid = (l+r)/2;
    buildst(l, mid, idx*2);
    buildst(mid+1, r, idx*2+1);

    st[idx] = max(st[idx*2], st[idx*2+1]);
```

```
}

ll query(int l, int r, int idx, int L, int R)  //L,R是操作的區間
{
    if(r < L || R < l) return -INF;
    if(L <= l && r <= R) return st[idx];
    int mid = (l+r)/2;
    return max(query(l, mid, idx*2, L, R), query(mid+1, r, idx*2+1, L, R));
}

void modify(int l, int r, int idx, int x, int v)
{
    if(r < x || x < l) return;
    if(l == r)
    {
        st[idx] += v; return;
    }
    int mid = (l+r)/2;
    modify(l, mid, idx*2, x, v);
    modify(mid+1, r, idx*2+1, x, v);
    st[idx] = max(st[idx*2], st[idx*2+1]);
}
```

1d segment tree + lazy tag

```
//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
struct Node //data = data*mul+add;
{
    ll data, mul, add;
};

ll getval(int l, int r, int idx)
{
    return (st[idx].data*st[idx].mul%MD+(r-l+1)*st[idx].add%MD)%MD;
}

void up(int l, int r, int idx)
{
    int mid = l+(r-l)/2;
    st[idx].data = (getval(l, mid, idx*2)+getval(mid+1, r, idx*2+1))%MD;
}

void down(int l, int r, int idx)
{
    st[idx].data = getval(l, r, idx);
    int lson = idx*2, rson = idx*2+1;
    if(l != r)
    {
        st[lson].mul = st[lson].mul*st[idx].mul%MD;
        st[lson].add = (st[lson].add*st[idx].mul+st[idx].add)%MD;
        st[rson].mul = st[rson].mul*st[idx].mul%MD;

        st[rson].add = (st[rson].add*st[idx].mul+st[idx].add)%MD;
```

```
    }
    st[idx].mul = 1, st[idx].add = 0;
}

void buildst(int l, int r, int idx)
{
    st[idx].mul = 1, st[idx].add = 0;
    if(l == r)
    {
        st[idx].data = arr[l];
        return;
    }
    int mid = l+(r-l)/2;
    buildst(l, mid, idx*2);
    buildst(mid+1, r, idx*2+1);
    up(l, r, idx);
}

void add(int l, int r, int idx, int L, int R, int v) //操作L,R
{
    if(r < L || R < l) return;
    if(L <= l && r <= R)
    {
        st[idx].add = (st[idx].add+v)%MD;
        return;
    }
    down(l, r, idx);
    int mid = l+(r-l)/2;
    add(l, mid, idx*2, L, R, v);
    add(mid+1, r, idx*2+1, L, R, v);
    up(l, r, idx);
}

void mul(int l, int r, int idx, int L, int R, int v)
{
    if(r < L || R < l) return;
    if(L <= l && r <= R)
    {
        st[idx].add = st[idx].add*v%MD;
        st[idx].mul = st[idx].mul*v%MD;
        return;
    }
    down(l, r, idx);
    int mid = l+(r-l)/2;
    mul(l, mid, idx*2, L, R, v);
    mul(mid+1, r, idx*2+1, L, R, v);
    up(l, r, idx);
}

ll query(int l, int r, int idx, int L, int R)
{
    if(r < L || R < l) return 0;

    if(L <= l && r <= R)
```

```
    {
        return getval(l, r, idx);
    }
    down(l, r, idx);
    int mid = l+(r-l)/2;
    return (query(l, mid, idx*2, L, R)+query(mid+1, r, idx*2+1, L, R))%MD;
}
```

binary index tree

```
#define lowbit(x) x&-x

int arr[N]; //紀錄前綴和
int bit[N];

void conv(int a[], int n)  //離散化
{
    vector<int> tmp;
    for(int i = 1; i <= n; i++) tmp.push_back(a[i]);
    sort(tmp.begin(), tmp.end());
    for(int i = 1; i <= n; i++) a[i] = lower_bound(tmp.begin(), tmp.end(), a[i]) -
tmp.begin() + 1;
}

void buildbit() //每個bit[x]紀錄[x-lowbit(x)+1, x]的總和
{
    for(int i = 0; i < n; i++) bit[i] = arr[i]-arr[i-lowbit(i)];
}

int sum(int x) //查詢[1,x]的總和
{
    int rtn = 0;
    for(;x;x-=lowbit(x)) rtn += bit[x];
    return rtn;
}

void modify(int x, int d) //把位置x的東西加上d
{
    for(;x<=n;x+=lowbit(x)) bit[x] += d;
}
```

2d segment tree

```
//二維陣列單點查詢區間加值
class St1d
{
private:
    ll st[4*N];

public:
    void build();
```

```
    void modify(int l, int r, int idx, int L, int R, ll v);
    ll query(int l, int r, int idx, int x);
    void down(int idx);
};

void St1d::build()
{
    memset(st, 0, sizeof(st));
}

void St1d::modify(int l, int r, int idx, int L, int R, ll v)
{
    if(r < L || R < l) return;
    if(L <= l && r <= R)
    {
        st[idx] += v;
        return;
    }
    assert(l != r);
    down(idx);
    int mid = (l+r)/2;
    modify(l, mid, idx*2, L, R, v);
    modify(mid+1, r, idx*2+1, L, R, v);
}

ll St1d::query(int l, int r, int idx, int x)
{
    if(x < l || r < x) return 0;
    if(l == x && r == x) return st[idx];
    down(idx);
    int mid = (l+r)/2;
    ll left = query(l, mid, idx*2, x);
    ll right = query(mid+1, r, idx*2+1, x);
    return left+right;
}

void St1d::down(int idx)
{
    st[idx*2] += st[idx], st[idx*2+1] += st[idx];
    st[idx] = 0;
}

//////////////////////

class St2d
{
private:
    St1d st[4*N];

public:
    void build(int il, int ir, int idx);


    void modify(int il, int ir, int jl, int jr, int idx, int iL, int iR, int jL, int
```

```
jR, ll v);
    ll query(int il, int ir, int jl, int jr, int idx, int i, int j);
};

void St2d::build(int il, int ir, int idx)
{
    st[idx].build();
    if(il == ir) return;
    int mid = (il+ir)/2;
    build(il, mid, idx*2);
    build(mid+1, ir, idx*2+1);
}

void St2d::modify(int il, int ir, int jl, int jr, int idx, int iL, int iR, int jL,
int jR, ll v)
{
    if(ir < iL || iR < il) return;
    if(iL <= il && ir <= iR)
    {
        st[idx].modify(jl, jr, 1, jL, jR, v); return;
    }
    int mid = (il+ir)/2;
    modify(il, mid, jl, jr, idx*2, iL, iR, jL, jR, v);
    modify(mid+1, ir, jl, jr, idx*2+1, iL, iR, jL, jR, v);
}

ll St2d::query(int il, int ir, int jl, int jr, int idx, int i, int j)
{
    ll tot = 0;
    if(i < il || ir < i) return 0;
    if(il <= i && i <= ir) tot += st[idx].query(jl, jr, 1, j);
    if(il == i && ir == i) return tot;
    int mid = (il+ir)/2;
    tot += query(il, mid, jl, jr, idx*2, i, j);
    tot += query(mid+1, ir, jl, jr, idx*2+1, i, j);
    return tot;
}
```

merge spilt treap

```
struct Treap
{
    int pri, sz;
    int rev;
    ll data, sum;   // tag: make-same
    Treap *lchild, *rchild;
    Treap(ll d):pri(rand()), sz(1), rev(0), data(d), sum(d), lchild(NULL),
rchild(NULL)
    {
    }
    inline void up();

    inline void down();
```

```
};

inline int size(Treap *t) { return t? t->sz:0; }
inline ll get_data(Treap *t) { return t? t->data:0; }
inline ll get_sum(Treap *t) { return t? t->sum:0; }

inline void Treap::up()
{
    if(lchild) lchild->down();
    if(rchild) rchild->down();
    sz = 1+size(lchild)+size(rchild);
    sum = get_sum(lchild) + data + get_sum(rchild);
}

inline void Treap::down()
{
    if(rev)
    {
        swap(mxpre, mxpost);
        swap(lchild, rchild);
        if(lchild) lchild->rev ^= 1;
        if(rchild) rchild->rev ^= 1;
        rev ^= 1;
    }
}

Treap *merge(Treap *a, Treap *b)
{
    if(!a || !b) return (a? a:b);
    if(a->pri < b->pri)
    {
        a->down();
        a->rchild = merge(a->rchild, b);
        a->up();
        return a;
    }
    else
    {
        b->down();
        b->lchild = merge(a, b->lchild);
        b->up();
        return b;
    }
}

void split(Treap *o, Treap *&a, Treap *&b, int k)
{
    if(!o) a = b = NULL;
    else
    {
        o->down();
        if(k >= size(o->lchild)+1)

        {
```

```
            a = o;
            split(o->rchild, a->rchild, b, k-size(o->lchild)-1);
        }
        else
        {
            b = o;
            split(o->lchild, a, b->lchild, k);
        }
        o->up();
    }
}
```

heavy light decomposition

```
#include<vector>
#define MAXN 100005
typedef std::vector<int >::iterator VIT;
int siz[MAXN],max_son[MAXN],pa[MAXN],dep[MAXN];
/*節點大小、節點大小最大的孩子、父母節點、深度*/
int link_top[MAXN],link[MAXN],cnt;
/*每個點所在鏈的鏈頭、樹鏈剖分的DFS序、時間戳*/
std::vector<int >G[MAXN];/*用vector存樹*/
void find_max_son(int x){
    siz[x]=1;
    max_son[x]=-1;
    for(VIT i=G[x].begin();i!=G[x].end();++i){
        if(*i==pa[x])continue;
        pa[*i]=x;
        dep[*i]=dep[x]+1;
        find_max_son(*i);
        if(max_son[x]==-1||siz[*i]>siz[max_son[x]])max_son[x]=*i;
        siz[x]+=siz[*i];
    }
}
void build_link(int x,int top){
    link[x]=++cnt;/*記錄x點的時間戳*/
    link_top[x]=top;
    if(max_son[x]==-1)return;
    build_link(max_son[x],top);/*優先走訪最大孩子*/
    for(VIT i=G[x].begin();i!=G[x].end();++i){
        if(*i==max_son[x]||*i==pa[x])continue;
        build_link(*i,*i);
    }
}
inline int find_lca(int a,int b){
    /*求LCA，可以在過程中對區間進行處理*/
    int ta=link_top[a],tb=link_top[b];
    while(ta!=tb){
        if(dep[ta]<dep[tb]){
            std::swap(ta,tb);
            std::swap(a,b);

        }
```

```
        //這裡可以對a所在的鏈做區間處理
        //區間為(link[ta],link[a])
        ta=link_top[a=pa[ta]];
    }
    /*最後a,b會在同一條鏈，若a!=b還要在進行一次區間處理*/
    return dep[a]<dep[b]?a:b;
}
```

Graph

minimum spanning tree (kruskal)

```
struct edge { int u, v, cost; };

bool comp(const edge& e1, const edge& e2)
{
    return e1.cost < e2.cost;
}

int kruskal()
{
    sort(es, es + e, comp);
    dset s(v);
    int res = 0;
    for(int i = 0; i < e; i++)
    {
        edge E = es[i];
        if(s.Find(E.u) != s.Find(E.v))
        {
            s.Union(E.u, E.v);
            res += E.cost;
        }
    }
    return res;
}
```

minimum spanning tree (prim)

```
int cost[100][100];
bool used[100];
int mincost[100];
int v, e;
#define INF 2147483647

int prim()
{
    for(int i = 0; i < v; i++)
    {
        mincost[i] = INF;

        used[i] = false;
```

```
    }
    mincost[0] = 0;
    int res = 0;

    while(true)
    {
        int x = -1;
        for(int u = 0; u < v; u++)
            if(!used[u] && (x == -1 || mincost[u] < mincost[x])) x = u;

        if(x == -1) break;
        used[x] = true;
        res += mincost[x];

        for(int u = 0; u < v; u++)
            mincost[u] = min(mincost[u], cost[x][u]);
    }
    return res;
}

void init()
{
    for(int i = 0; i < v; i++)
        for(int j = 0; j < v; j++)
            if(i == j) cost[i][j] = 0;
            else cost[i][j] = INF;
}
```

shortest path (floyd)

```
int d[N][N];

void init()
{
    for(int i = 0; i < v; i++)
        for(int j = 0; j < v; j++)
            if(i == j) d[i][j] = 0;
            else d[i][j] = INF;
}

void floyd_warshall()
{
    for(int k = 0; k < v; k++)
        for(int i = 0; i < v; i++)
            for(int j = 0; j < v; j++)
                if(d[i][k] != INF && d[k][j] != INF)
                    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
```

shortest path (dijkstra)

```cpp
struct edge{int to, cost;};
typedef pair<int, int> P; //first = min distance, second = v id
#define f first
#define s second
#define INF 2147483647

int V, E, S, F;
vector<edge> G[100];
int d[100];

void dijkstra()
{
    priority_queue<P, vector<P>, greater<P>> q;
    fill(d, d + V, INF);
    d[S] = 0;
    q.push(P(0, S));

    while(!q.empty())
    {
        P p = q.top(); q.pop();
        int v = p.s;
        if(d[v] < p.f) continue;
        for(int i = 0; i < G[v].size(); i++)
        {
            edge e = G[v][i];
            if(d[e.to] > d[v] + e.cost)
            {
                d[e.to] = d[v] + e.cost;
                q.push(P(d[e.to], e.to));
            }
        }
    }
}
```

shortest path (bellman)

```cpp
int d[N][N];

void init()
{
    for(int i = 0; i < v; i++)
        for(int j = 0; j < v; j++)
            if(i == j) d[i][j] = 0;
            else d[i][j] = INF;
}

void floyd_warshall()
{
    for(int k = 0; k < v; k++)
        for(int i = 0; i < v; i++)
            for(int j = 0; j < v; j++)
                if(d[i][k] != INF && d[k][j] != INF)
```

```
                    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}
```

shortest path (spfa)

```
typedef pair<int, ll> P;
#define idx first
#define w second
int vn, en;
vector<P> graph[N];
ll dist[N];

bool spfa()  // return true if neg cycle
{
    for(int i = 0; i < vn; i++) dist[i] = INF; dist[0] = 0;
    int cnt[N] = {0};
    bool inq[N] = {false};
    queue<int> q; q.push(0); inq[0] = true;
    while(!q.empty())
    {
        int s = q.front(); q.pop();
        inq[s] = false;
        for(auto e:graph[s])
        {
            if(dist[e.idx] > dist[s]+e.w)
            {
                dist[e.idx] = dist[s]+e.w;
                if(++cnt[e.idx] >= vn) return true;
                if(!inq[e.idx])
                {
                    inq[e.idx] = true;
                    q.push(e.idx);
                }
            }
        }
    }
    return false;
}
```

Flow

Maximum Flow

```
template<typename T>
struct DINIC{
    static const int MAXN=105;
    static const T INF=INT_MAX;
    int n, level[MAXN], cur[MAXN];
    struct edge{

        int v,pre;
```

```
        T cap,flow,r;
        edge(int v,int pre,T cap):v(v),pre(pre),cap(cap),flow(0),r(cap){}
    };
    int g[MAXN];
    vector<edge> e;
    void init(int _n){
        memset(g,-1,sizeof(int)*((n=_n)+1));
        e.clear();
    }
    void add_edge(int u,int v,T cap,bool directed=false){
        e.push_back(edge(v,g[u],cap));
        g[u]=e.size()-1;
        e.push_back(edge(u,g[v],directed?0:cap));
        g[v]=e.size()-1;
    }
    int bfs(int s,int t){
        memset(level,0,sizeof(int)*(n+1));
        memcpy(cur,g,sizeof(int)*(n+1));
        queue<int> q;
        q.push(s);
        level[s]=1;
        while(q.size()){
            int u=q.front();q.pop();
            for(int i=g[u];~i;i=e[i].pre){
                if(!level[e[i].v]&&e[i].r){
                    level[e[i].v]=level[u]+1;
                    q.push(e[i].v);
                    if(e[i].v==t)return 1;
                }
            }
        }
        return 0;
    }
    T dfs(int u,int t,T cur_flow=INF){
        if(u==t)return cur_flow;
        T df;
        for(int &i=cur[u];~i;i=e[i].pre){
            if(level[e[i].v]==level[u]+1&&e[i].r){
                if(df=dfs(e[i].v,t,min(cur_flow,e[i].r))){
                    e[i].flow+=df;
                    e[i^1].flow-=df;
                    e[i].r-=df;
                    e[i^1].r+=df;
                    return df;
                }
            }
        }
        return level[u]=0;
    }
    T dinic(int s,int t,bool clean=true){
        if(clean){
            for(size_t i=0;i<e.size();++i){

                e[i].flow=0;
```

```
                e[i].r=e[i].cap;
            }
        }
        T ans=0, mf=0;
        while(bfs(s,t))while(mf=dfs(s,t))ans+=mf;
        return ans;
    }
};
```

Minimum Cost Flow

```
template<typename _T>
struct MCMF{
    static const int MAXN=440;
    static const _T INF=999999999;
    struct edge{
        int v,pre;
        _T cap,cost;
        edge(int v,int pre,_T cap,_T cost):v(v),pre(pre),cap(cap),cost(cost){}
    };
    int n,S,T;
    _T dis[MAXN],piS,ans;
    bool vis[MAXN];
    vector<edge> e;
    int g[MAXN];
    void init(int _n){
        memset(g,-1,sizeof(int)*((n=_n)+1));
        e.clear();
    }
    void add_edge(int u,int v,_T cap,_T cost,bool directed=false){
        e.push_back(edge(v,g[u],cap,cost));
        g[u]=e.size()-1;
        e.push_back(edge(u,g[v],directed?0:cap,-cost));
        g[v]=e.size()-1;
    }
    _T augment(int u,_T cur_flow){
        if(u==T||!cur_flow)return ans+=piS*cur_flow,cur_flow;
        vis[u]=1;
        _T r=cur_flow,d;
        for(int i=g[u];~i;i=e[i].pre){
            if(e[i].cap&&!e[i].cost&&!vis[e[i].v]){
                d=augment(e[i].v,min(r,e[i].cap));
                e[i].cap-=d;
                e[i^1].cap+=d;
                if(!(r-=d))break;
            }
        }
        return cur_flow-r;
    }
    bool modlabel(){
        for(int u=0;u<=n;++u)dis[u]=INF;

        static deque<int>q;
```

```
        dis[T]=0,q.push_back(T);
        while(q.size()){
            int u=q.front();q.pop_front();
            _T dt;
            for(int i=g[u];~i;i=e[i].pre){
                if(e[i^1].cap&&(dt=dis[u]-e[i].cost)<dis[e[i].v]){
                    if((dis[e[i].v]=dt)<=dis[q.size()?q.front():S]){
                        q.push_front(e[i].v);
                    }else q.push_back(e[i].v);
                }
            }
        }
        for(int u=0;u<=n;++u)
            for(int i=g[u];~i;i=e[i].pre)
                e[i].cost+=dis[e[i].v]-dis[u];
        return piS+=dis[S], dis[S]<INF;
    }
    _T mincost(int s,int t){
        S=s,T=t;
        piS=ans=0;
        while(modlabel()){
            do memset(vis,0,sizeof(bool)*(n+1));
            while(augment(S,INF));
        }return ans;
    }
};
```

Divide and Conquer

樹重心分治

```
void cal_subsz(int v, int p)
{
    ll s = 1;
    for(int i = 0; i < graph[v].size(); i++)
    {
        int c = graph[v][i].idx;
        if(c == p || iscentroid[c]) continue;
        cal_subsz(c, v);
        s += subsz[c];
    }
    subsz[v] = s;
}

Edge find_centroid(int v, int p, const ll sz)
{
    Edge cen(-1, INF);
    ll mxsz = -1;
    for(int i = 0; i < graph[v].size(); i++)
    {
        int c = graph[v][i].idx;
        if(c == p || iscentroid[c]) continue;
```

```
        Edge res = find_centroid(c, v, sz);
        if(res.w < cen.w) cen = res;
        mxsz = max(mxsz, subsz[c]);
    }
    mxsz = max(mxsz, sz-subsz[v]);
    if(mxsz < cen.w) cen.idx = v, cen.w = mxsz;
    return cen;
}

void get_dist(int v, int p, ll w)
{
    dist.push_back(w);
    for(int i = 0; i < graph[v].size(); i++)
    {
        int c = graph[v][i].idx;
        if(c == p || iscentroid[c]) continue;
        get_dist(c, v, w+graph[v][i].w);
    }
}

ll cal_pair(int idx, ll w)
{
    dist.clear();
    get_dist(idx, -1, w);
    sort(dist.begin(), dist.end());
    ll sum = 0;
    for(int l = 0, r = dist.size()-1; l < r; )
    {
        if(dist[r]+dist[l] <= k) sum += r-l, l++;
        else r--;
    }
    return sum;
}

ll tree_dc(int v)
{
    ll sum = 0;
    // find centroid
    cal_subsz(v, -1);
    int centroid = find_centroid(v, -1, subsz[v]).idx;
    iscentroid[centroid] = true;
    // cal result
    sum += cal_pair(centroid, 0);
    // for ever subtree
    for(int i = 0; i < graph[centroid].size(); i++)
    {
        int c = graph[centroid][i].idx; ll w = graph[centroid][i].w;
        if(iscentroid[c]) continue;
        // cal res
        sum -= cal_pair(c, w);
        // dc
        sum += tree_dc(c);

    }
```

```
    return sum;
}
```

求最近點對距離

```
bool cmp_y(P &a, P &b)  // y increasing
{
    return a.y < b.y;
}

bool cmp_x(P &a, P &b) // x increasing
{
    return a.x < b.x;
}

void init()
{
    sort(arr, arr+n, cmp_y);
}

double nearestDist(int l, int r)
{
    if(l == r) return (double)INT_MAX;
    int mid = (l+r)/2;
    double d = min(dc(l, mid), dc(mid+1, r));
    sort(arr+l, arr+r+1, cmp_x);
    double center = arr[mid].x;
    vector<P> cen;
    for(int i = l; i <= r; i++)
    {
        if(fabs(arr[i].x-center) >= d) continue;
        for(auto p:cen)
        {
            double dx = fabs(arr[i].x-p.x), dy = fabs(arr[i].y-p.y);
            if(dy < d)
            {
                d = min(d, sqrt(dx*dx+dy*dy));
            }
        }
        cen.push_back(arr[i]);
    }
    return d;
}
```

String

KMP

```
void failure_build(const char *p, int *fail)

{
```

```
    for(int i=1, j=fail[0]=-1; p[i]; i++)
    {
        while(j>=0&&p[j+1]!=p[i]) j=fail[j];
        if(p[j+1]==p[i]) j++;
        fail[i]=j;
    }
}
int KMP(const char *T, const char *P, int *fail)
{
    failure_build(P, fail);
    for(int i=0, j=-1; T[i]; i++)
    {
        while(j>=0&&P[j+1]!=T[i]) j=fail[j];
        if(P[j+1]==T[i]) j++;
        if(!P[j+1]) return i-j;
    }
    return -1;
}

//使用方法: KMP(主字串, 待匹配字串, failure array)
//回傳: 第一個完全匹配的位置
```

Z Value

```
void Z_build(const char *S, int *Z)
{
    Z[0]=0;
    int b=0;
    for(int i=1;S[i];i++)
    {
        if(Z[b]+b<i) Z[i]=0;
        else Z[i]=min(Z[b]+b-i,Z[i-b]);
        while(S[i+Z[i]]&&S[Z[i]]==S[i+Z[i]]) Z[i]++;
        if(Z[i]+i>Z[b]+b) b=i;
    }
}
```

Suffix Array

```
void SA_radix_sort(int *s, int *e, int *Rank, int rankcnt)
{
    int box[MAX_N], tmp[MAX_N], len=e-s;
    memset(box,0,sizeof(int)*rankcnt);
    for(int i=0;i<len;i++) box[Rank[i]]++;
    for(int i=1;i<rankcnt;i++) box[i]=box[i]+box[i-1];
    for(int i=len-1;i>=0;i--) tmp[--box[Rank[s[i]]]]=s[i];
    for(int i=0;i<len;i++) s[i]=tmp[i];
}
#define equal(a,b,c) c[a]!=c[b]||a+k>=len||c[a+k]!=c[b+k]
void SA_build(int *SA, int *Rank, char *S)
{
```

```
    int ranktmp[MAX_N], len=strlen(S), rankcnt='z'+1;
    for(int i=0;i<len;i++) Rank[i]=S[i];
    for(int k=1;rankcnt!=len;k*=2)
    {
        for(int i=0;i<len;i++) SA[i]=(i+len-k)%len;
        SA_radix_sort(SA+k, SA+len, Rank+k, rankcnt);
        SA_radix_sort(SA, SA+len, Rank, rankcnt);
        ranktmp[SA[0]]=0, rankcnt=0;
        for(int i=1;i<len;i++)
            ranktmp[SA[i]]=rankcnt+=equal(SA[i-1], SA[i], Rank);
        rankcnt++;
        for(int i=0;i<len;i++) Rank[i]=ranktmp[i];
    }
}
#undef equal
```

Suffix Array + STL

```
struct CMP
{
    int len,k,*Rank,a,b;
    inline bool operator()(int i, int j)
    {
        if(Rank[i]!=Rank[j])return Rank[i]<Rank[j];
        a=(i+=k)<len?Rank[i]:-1;
        b=(j+=k)<len?Rank[j]:-1;
        return a<b;
    }
};
void SA_build(int *SA, int *Rank, char *S){
    int tmp[MAX_N], len=strlen(S);
    for(int i=0;i<len;i++) SA[i]=i, Rank[i]=S[i];
    CMP cmp={len,1};
    while(cmp.k*=2)
    {
        cmp.Rank=Rank;
        sort(SA,SA+len,cmp);
        tmp[SA[0]]=0;
        for(int i=1;i<len;i++)
            tmp[SA[i]]=tmp[SA[i-1]]+cmp(SA[i-1],SA[i]);
        if(tmp[SA[len-1]]==len-1) break;
        for(int i=0;i<len;i++) Rank[i]=tmp[i];
    }
}
```

LCP

```
//build query in O(nlogn), query LCP(i,j) in O(1)
int dp_height[MAX_N][20];
void height_build(int *SA, int *Rank, char *S, int *Height)
{
```

```
    int len=strlen(S), k=0;
    for(int i=0;i<len;i++)
    {
        if(Rank[i]==0) continue;
        while(S[i+k] == S[SA[Rank[i]-1]+k]) k++;
        Height[Rank[i]]=k;
        if(k) k--;
    } Height[0]=0;
    for(int i=0;i<len;i++) dp_height[i][0]=Height[i];
    for(int i=0;i<len;i++) for(int j=1;i+(1<<j)<len;j++)
        dp_height[i][j]=min(dp_height[i][j-1], dp_height[i+(1<<(j-1))][j-1]);
}
int height_query(int x, int y)
{
    int k=0;
    while((1<<(k+1))<=y-x) k++;
    return min(dp_height[x+1][k], dp_height[y-(1<<k)+1][k]);
}
```