字符串双哈希

```
typedef pair<int,int> hashv;
const LL mod1=1000000007;
const LL mod2=1000000009;

hashv operator + (hashv a, hashv b) {
    int c1=a.fi+b.fi,c2=a.se+b.se;
    if (c1>=mod1) c1-=mod1;
    if (c2>=mod2) c2-=mod2;
    return mkp(c1,c2);
}

hashv operator - (hashv a, hashv b) {
    int c1=a.fi-b.fi,c2=a.se-b.se;
    if (c1<0) c1+=mod1;
    if (c2<0) c2+=mod2;
    return mkp(c1,c2);
}

hashv operator * (hashv a, hashv b) {
    return mkp(1ll*a.fi*b.fi%mod1,1ll*a.se*b.se%mod2);
}

hashv pw[N], Hx[N];

hashv base=mkp(13331, 23333);

pw[0] = mkp(1,1);
for (int i=1;i<=n;i++)
    pw[i] = pw[i-1]*base, Hx[i] = Hx[i-1]*base+mkp(s[i],s[i]);

// 前面的才是高位，需要抵消掉
hashv hx1 = Hx[r]-Hx[l-1]*pw[r-l+1];
hashv hx2 = Hx[x]-Hx[x-len]*pw[len];

if (s1==s2) {
    // 双哈希相等
}
```

两球体积的交

```
double GetVol(P a, P b) {
    double dist2 = GetDist2(a, b);
    if(dist2 >= a.r+b.r) return 0;

    if(dist2+a.r <= b.r) return 4.0/3.0*PI*a.r*a.r*a.r;
    if(dist2+b.r <= a.r) return 4.0/3.0*PI*b.r*b.r*b.r;

    /*
    double dx = (a.r*a.r-b.r*b.r)/dist2;
    double L = (dist2+dx)/2.0, l = dist2-L;
    double x1 = a.r - L, x2 = b.r - l;
    double res = PI*x1*x1*(a.r - x1 / 3.0);
    res += PI*x2*x2*(b.r - x2 / 3.0);
    */

    double cosA = (a.r*a.r+dist2*dist2-b.r*b.r)/(2.0*dist2*a.r);
    double hA = a.r*(1.0-cosA);
    double res = PI*hA*hA*(3.0*a.r-hA)/3.0;

    double cosB = (b.r*b.r+dist2*dist2-a.r*a.r)/(2.0*dist2*b.r);
    double hB = b.r*(1.0-cosB);
    res += PI*hB*hB*(3.0*b.r-hB)/3.0;

    return res;
}
```

凸壳优化

```
using T = __int128;
struct Point {
    T x;
    T y;
    Point(T x = 0, T y = 0) : x(x), y(y) {}

    Point &operator+=(const Point &p) {
        x += p.x, y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) {
        x -= p.x, y -= p.y;
        return *this;
    }
    Point &operator*=(const T &v) {
        x *= v, y *= v;
        return *this;
    }
    friend Point operator-(const Point &p) {
        return Point(-p.x, -p.y);
    }
    friend Point operator+(Point lhs, const Point &rhs) {
        return lhs += rhs;
    }
    friend Point operator-(Point lhs, const Point &rhs) {
        return lhs -= rhs;
    }
    friend Point operator*(Point lhs, const T &rhs) {
        return lhs *= rhs;
    }
};

T dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}

T cross(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}
```

Dinic

```
int idx, h[N], ne[M], ver[M], e[M];
int n, m, S, T;
int d[N], cur[N];

void add(int a, int b, int c) {
    ver[idx] = b, e[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}
bool bfs() {
    queue<int> q;
    memset(d, -1, sizeof d);
    q.push(S);
    d[S] = 0, cur[S] = h[S];
    while(q.size()) {
        int t = q.front(); q.pop();
        for(int i = h[t]; ~i; i = ne[i]) {
            int v = ver[i];
            if(d[v] == -1 && e[i]) {
                d[v] = d[t]+1;
                cur[v] = h[v];
                if(v == T) return true;
                q.push(v);
            }
        }
    }
    return false;
}
int update(int u, int limit) {
    if(u == T) return limit;
    int flow = 0;
    for(int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int v = ver[i];
        if(d[v] == d[u]+1 && e[i]) {
            int t = update(v, min(e[i], limit-flow));
            if(!t) d[v] = -1;
            flow += t;
            e[i] -= t;
            e[i^1] += t;
        }
    }
    return flow;
}
int dinic() {
    int res = 0, flow;
    while(bfs())
        while(flow = update(S, INF))
            res += flow;
    return res;
}
```

线段树 & LCA

```
void BuildPhiTree(int n) {
    dep[1] = 1;
```

```cpp
    for(int k = 0; k < 6; k ++) fa[1][k] = 1;
    for(int i = 2; i <= n; i ++) {
        dep[i] = dep[phi[i]] + 1;
        fa[i][0] = phi[i];
        for(int k = 1; k < 6; k ++)
            fa[i][k] = fa[fa[i][k-1]][k-1];
    }
}
int LCA(int x, int y) {
    if(dep[x] < dep[y]) swap(x, y);
    for(int k = 5; k >= 0; k --)
        if(dep[fa[x][k]] >= dep[y])
            x = fa[x][k];
    if(x == y) return x;
    for(int k = 5; k >= 0; k --)
        if(fa[x][k] != fa[y][k])
            x = fa[x][k], y = fa[y][k];
    return fa[x][0];
}
struct Info {
    int l, r;
    int lca, ans;
    bool root;
};
struct SegTree {
    Info tr[N<<2];
    #define ls(u) (u<<1)
    #define rs(u) (u<<1|1)

    void PushUp(int u) {
        tr[u].lca = LCA(tr[ls(u)].lca, tr[rs(u)].lca);
        tr[u].ans = tr[ls(u)].ans + tr[rs(u)].ans +
                    (tr[ls(u)].r-tr[ls(u)].l+1) * (dep[tr[ls(u)].lca]-
dep[tr[u].lca]) +
                    (tr[rs(u)].r-tr[rs(u)].l+1) * (dep[tr[rs(u)].lca]-
dep[tr[u].lca]);
        tr[u].root = tr[ls(u)].root && tr[rs(u)].root;
    }
    void Build(int u, int l, int r) {
        if(l == r) {
            tr[u] = {l, r, a[r], 0, a[r] == 1};
            return;
        }
        tr[u] = {l, r};
        int mid = l+r >> 1;
        Build(ls(u), l, mid), Build(rs(u), mid+1, r);
        PushUp(u);
    }
    void Modify(int u, int l, int r) {
        if(tr[u].root) return;
        if(tr[u].l == tr[u].r) {
            tr[u].lca = fa[tr[u].lca][0];
            tr[u].root = tr[u].lca == 1;
            return;
        }
        int mid = tr[u].l+tr[u].r >> 1;
        if(l <= mid) Modify(ls(u), l, r);
        if(r > mid) Modify(rs(u), l, r);
        PushUp(u);
    }
    int QueryLCA(int u, int l, int r) {
        if(l <= tr[u].l && tr[u].r <= r) return tr[u].lca;
        int mid = tr[u].l+tr[u].r >> 1;
        int lca1 = 0, lca2 = 0;
        if(l <= mid) lca1 = QueryLCA(ls(u), l, r);
        if(r > mid) lca2 = QueryLCA(rs(u), l, r);
        // cout << tr[u].l << ' ' << tr[u].r << " : " << lca1 << ' ' << lca2 <<
' ' << LCA(lca1, lca2) << '\n';
        if(!lca1) return lca2;
        if(!lca2) return lca1;
        return LCA(lca1, lca2);
    }
    int QueryAns(int u, int l, int r, int lca) {
        if(l <= tr[u].l && tr[u].r <= r) return tr[u].ans + (tr[u].r-tr[u].l+1)*
(dep[tr[u].lca]-dep[lca]);
        int mid = tr[u].l+tr[u].r >> 1;
        int ans = 0;
        if(l <= mid) ans += QueryAns(ls(u), l, r, lca);
        if(r > mid) ans += QueryAns(rs(u), l, r, lca);
        return ans;
    }
    #undef ls
    #undef rs
}sgt;
```

DSU

```
struct DSU {// int / LL
    vector<int> fa, siz;

    void Init(int n) {
        fa.resize(n+1);
        siz.resize(n+1);
        for(int i = 1; i <= n; i ++)
            fa[i] = i, siz[i] = 1;
    }
    int GetFa(int x) {
        if(x == fa[x]) return x;
        return fa[x] = GetFa(fa[x]);
    }
    bool Same(int x, int y) {
        return GetFa(x) == GetFa(y);
    }
    bool Merge(int x, int y) {
        x = GetFa(x), y = GetFa(y);
        if(x == y) return false;
        siz[x] += siz[y];
        fa[y] = x;
        return true;
    }
    int Size(int x) {
        return siz[GetFa(x)];
    }
}dsu;
```

Fenwick Tree

```
struct FenwickTree { // int / LL
    int n;
    vector<int> a;
    void Init(int n) {
        this->n = n;
        a.assign(n+1, 0);
    }
    int Lowbit(int x) {
        return x & -x;
    }
    void Add(int x, int v) {
        for( ; x <= n; x += Lowbit(x))
            a[x] += v;
    }
    int Sum(int x) {
        int ans = 0;
        for( ; x; x -= Lowbit(x))
            ans += a[x];
        return ans;
    }
    int RangeSum(int l, int r) {
        return Sum(r) - Sum(l-1);
    }
}fen;
```

复数

```
struct Complex{
    double x, y;
    Complex(double xx = 0.0, double yy = 0.0) {
        x = xx, y = yy;
    }
    Complex operator + (const Complex &o) const {
        return Complex(x+o.x, y+o.y);
    }
    Complex operator - (const Complex &o) const {
        return Complex(x-o.x, y-o.y);
    }
    Complex operator * (const Complex &o) const {
        return Complex(x*o.x-y*o.y, x*o.y+y*o.x);
    }
};
```

FFT

```
const double Pi = acos(-1.0);

struct Complex{
    double x, y;
    Complex(double xx = 0.0, double yy = 0.0) {
        x = xx, y = yy;
    }
    Complex operator + (const Complex &o) const {
        return Complex(x+o.x, y+o.y);
    }
    Complex operator - (const Complex &o) const {
        return Complex(x-o.x, y-o.y);
    }
    Complex operator * (const Complex &o) const {
        return Complex(x*o.x-y*o.y, x*o.y+y*o.x);
    }
};

int n, m, limit = 1;
Complex a[N], b[N];

// 递归版本
void FFT(int tlimit, Complex a[], int on) {
    if(tlimit == 1) return;
    Complex a1[tlimit>>1], a2[tlimit>>1];
    for(int i = 0; i < tlimit; i += 2)
        a1[i>>1] = a[i], a2[i>>1] = a[i+1];
    FFT(tlimit>>1, a1, on), FFT(tlimit>>1, a2, on);
    Complex Wn(cos(2.0*Pi/tlimit), on*sin(2.0*Pi/tlimit)), w(1, 0);
    for(int i = 0; i < (tlimit>>1); i ++, w = w*Wn) {
        auto t = w * a2[i];
        a[i] = a1[i] + t;
        a[i+(tlimit>>1)] = a1[i] - t;
    }
}
// 迭代版本 Faster
void FFT(Complex a[], int on) {
    for(int i = 0; i < limit; i ++)
        if(i < R[i]) swap(a[i], a[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        Complex Wn(cos(Pi/mid), on*sin(Pi/mid));
        for(int j = 0, r = mid<<1; j < limit; j += r) {
            Complex w(1, 0);
            for(int k = 0; k < mid; k ++, w = w*Wn) {
                Complex x = a[j+k], y = w*a[j+mid+k];
                a[j+k] = x+y;
                a[j+mid+k] = x-y;
            }
        }
    }
}

void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i ++)
        cin >> a[i].x;
    for(int i = 0; i <= m; i ++)
        cin >> b[i].x;
    limit = 1;
    while(limit <= n+m) limit <<= 1;
    FFT(limit, a, 1), FFT(limit, b, 1);
    for(int i = 0; i < limit; i ++)
        a[i] = a[i]*b[i];
    FFT(limit, a, -1);
    for(int i = 0; i <= n+m; i ++){
        cout << (int)(a[i].x/limit+0.5) << " \n"[i == n+m];
    }
}
```

NTT

```
const int N = 4e6+10, MO = 998244353, G = 3, Gi = 332748118;

int n, m, limit = 1, rr;
int R[N];
int a[N], b[N];

int qpm(int a, int b, const int &c = MO) {
    int ans = 1;
    while(b) {
        if(b & 1) ans = 1LL*ans*a%MO;
        a = 1LL*a*a%MO;
        b >>= 1;
    }
    return ans;
}
void NTT(int a[], int on) {
    for(int i = 0; i < limit; i ++)
        if(i < R[i]) swap(a[i], a[R[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        int Wn = qpm((on == 1 ? G : Gi), (MO-1)/(mid<<1));
        for(int j = 0, r = mid<<1; j < limit; j += r) {
            int w = 1;
            for(int k = 0; k < mid; k ++, w = 1LL*w*Wn%MO) {
                int x = a[j+k], y = 1LL*w*a[j+mid+k]%MO;
                a[j+k] = (1LL*x+y)%MO;
                a[j+mid+k] = (1LL*x-y)%MO;
            }
        }
    }
}
void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i ++) {
        cin >> a[i];
        a[i] %= MO;
    }
    for(int i = 0; i <= m; i ++) {
        cin >> b[i];
        b[i] %= MO;
    }
    limit = 1, rr = 0;
    while(limit <= n+m) limit <<= 1, rr ++;
    for(int i = 0; i < limit; i ++)
        R[i] = (R[i>>1]>>1) | ((i&1)<<(rr-1));
    NTT(a, 1), NTT(b, 1);
    for(int i = 0; i <= limit; i ++)
        a[i] = 1LL*a[i]*b[i]%MO;
    NTT(a, -1);
    int inv = qpm(limit, MO-2);
    for(int i = 0; i <= n+m; i ++){
        cout << (1LL*a[i]*inv%MO+MO)%MO << " \n"[i == n+m];
    }
}
```