NOIP2018模板

前言

```
while(true) ++++++ ++++++ ++++++ RP;
```

快读快写

```
template <typename T> inline void read(T &x)
{
    char c; T tag = 1;
    while(!isdigit((c=getchar()))) if(c == '-') tag = -1;
    x = c-'0';
    while(isdigit((c=getchar()))) x = (x<<1)+(x<<3) + c-'0';
    x *= tag;
}

template <typename T> void write(T x)
{
    if(x < 0) x = -x, putchar('-');
    if(x > 9) write(x/10);
    putchar(x%10+'0');
}
```

```
ios::sync_with_stdio(false);
cin.tie(NULL);
```

堆

```
struct Heap
{
    static const int Maxn = 1e6+7;
    int sz, a[Maxn];
    Heap() { sz = 0; memset(a, 0, sizeof a); }
    inline bool cmp(int x, int y) { return x < y; } // 小根堆
    inline int size() { return sz; }
    inline bool empty() { return sz == 0; }
    inline int top() { return a[1]; }
    inline void push(int x) { a[++sz] = x; swift_up(sz); }
```

```
    inline void pop() { swap(a[1], a[sz--]); swift_down(1); }
    inline void swift_up(int p)
    {
        while(p > 1 && cmp(a[p], a[p>>1])) // a[p] < a[p<<1]
            swap(a[p], a[p>>1]), p >>= 1;
    }
    inline void swift_down(int p)
    {
        int l, r, s;
        while(true)
        {
            l = p<<1; r = p<<1|1;
            if(l > sz) break;
            if(r > sz || cmp(a[l], a[r])) s = l; // a[l] < a[r]
            else s = r;
            if(cmp(a[s], a[p])) // a[s] < a[p]
                swap(a[p], a[s]), p = s;
            else break;
        }
    }
};
```

字符串

manacher算法(回文字符串)

```
inline int manacher(const char *str, char *buf, int *p)
{
    int str_len = strlen(str), buf_len = 2;
    buf[0] = buf[1] = '#';
    for(int i = 0; i < str_len; ++i)
        buf[buf_len++] = str[i], buf[buf_len++] = '#';

    int mx = 0, id, ans = 0;
    for(int i = 1; i < buf_len; ++i)
    {
        if(i <= mx) p[i] = min(p[id*2-i], mx-i);
        else p[i] = 1;
        while(buf[i-p[i]] == buf[i+p[i]]) p[i]++;
        if(i+p[i] > mx) mx = i+p[i], id = i;
        ans = max(ans, p[i]-1);
    }
    return ans;
}
```

KMP

```
len_a = strlen(a+1);
```

```
len_b = strlen(b+1);
for(int i = 2, k = 0; i <= len_b; ++i)
{
    while(k && b[k+1] != b[i]) k = _next[k];
    if(b[k+1] == b[i]) ++k;
    _next[i] = k;
}
for(int i = 1, j = 0; i <= len_a; ++i)
{
    while(j && a[i] != b[j+1]) j = _next[j];
    if(b[j+1] == a[i]) ++j;
    if(j == len_b)
    {
        cout << i-len_b+1 << endl;
        j = _next[j];
    }
}
```

字符串哈希

```
inline unsigned long long _hash(const string &s)
{
    unsigned long long res = 0;
    for(int i = 0; i < s.length(); ++i)
        res = (res*Base+s[i])%Mod+Prime;
    return res;
}
```

快排

```
void quick_sort(int l, int r)
{
    if(l >= r) return;
    swap(a[l], a[l+rand()%(r-l)]);
    int i = l, j = r, mid = a[l];
    while(i < j)
    {
        while(i < j && a[j] >= mid) --j;
        swap(a[i], a[j]);
        while(i < j && a[i] < mid) ++i;
        swap(a[i], a[j]);
    }
    quick_sort(l, i-1);
    quick_sort(i+1, r);
}
```

求第K大数

HDOJ 2665 POJ 2104

```
int kth_element(int l, int r, int k)
{
    if(l == r) return a[l];
    swap(a[l], a[l+rand()%(r-l)]);
    int mid = a[l], i = l, j = r;
    while(i < j)
    {
        while(i < j && a[j] >= mid) --j;
        swap(a[i], a[j]);
        while(i < j && a[i] < mid) ++i;
        swap(a[i], a[j]);
    }
    a[i] = mid;
    if(i == k) return mid;
    else if(i > k) return kth_element(l, i-1, k);
    else return kth_element(i+1, r, k);
}
```

STL (排序,无返回值)

```
nth_element(a+1, a+k+1, a+n+1);
```

求逆序对(归并排序)

```
void merge_sort(int l, int r)
{
    if(l == r) return;
    int mid = (l+r)>>1;
    merge_sort(l, mid);
    merge_sort(mid+1, r);
    int i = l, j = mid+1, k = l;
    while(k <= r)
    {
        if(j <= r && (i > mid || a[j] < a[i]))
        {
            ans += mid-i+1;
            b[k++] = a[j++];
        }
        else b[k++] = a[i++];
    }
    memcpy(a+l, b+l, sizeof(int)*(r-l+1));
}
```

树

树的重心

```
void treedp(int cur, int fa)
{
    s[cur] = c[cur];
    for(int i = fir[cur]; i; i = nex[i])
    {
        if(e[i] == fa) continue;
        treedp(e[i], cur);
        s[cur] += s[e[i]];
        maxs[cur] = max(maxs[cur], s[e[i]]);
    }
    maxs[cur] = max(maxs[cur], sum-s[cur]);
}
```

二叉查找树

平衡树

线段树

区间修改区间查询

```
struct Tree
{
    int l, r;
    long long sum, add;
    Tree(){ sum = add = 0; }
} tr[Maxn<<2];

inline int ls(int x) { return x<<1; }
inline int rs(int x) { return x<<1|1; }
inline void push_up(int);
inline void push_down(int);
inline void build_tree(int, int, int);
inline void update_tree(int, int, int, long long);
inline long long query_tree(int, int, int);

inline void push_up(int i)
{
    tr[i].sum = tr[ls(i)].sum+tr[rs(i)].sum;
}

inline void push_down(int i)
{
    if(!tr[i].add) return;
    int len = tr[i].r-tr[i].l+1;
```

```
    tr[ls(i)].add += tr[i].add;
    tr[rs(i)].add += tr[i].add;
    tr[ls(i)].sum += tr[i].add*(len-len/2);
    tr[rs(i)].sum += tr[i].add*(len/2);
    tr[i].add = 0;
}

inline void build_tree(int i, int l, int r)
{
    tr[i].l = l; tr[i].r = r;
    if(l == r) { tr[i].sum = a[l]; return; }
    int mid = (l+r)>>1;
    build_tree(ls(i), l, mid);
    build_tree(rs(i), mid+1, r);
    push_up(i);
}

inline void update_tree(int i, int l, int r, long long k)
{
    if(tr[i].l >= l && tr[i].r <= r)
    {
        tr[i].sum += k*(tr[i].r-tr[i].l+1);
        tr[i].add += k;
        return;
    }
    push_down(i);
    int mid = (tr[i].l+tr[i].r)>>1;
    if(l <= mid) update_tree(ls(i), l, r, k);
    if(r > mid)  update_tree(rs(i), l, r, k);
    push_up(i);
}

inline long long query_tree(int i, int l, int r)
{
    if(tr[i].l >= l && tr[i].r <= r) return tr[i].sum;
    push_down(i);
    int mid = (tr[i].l+tr[i].r)>>1;
    long long res = 0;
    if(l <= mid) res += query_tree(ls(i), l, r);
    if(r > mid)  res += query_tree(rs(i), l, r);
    return res;
}
```

树状数组

单点修改区间查询 区间修改单点查询

```
// 单点修改 add(x, k);
// 区间修改 add(x, k); add(y+1, -k);
inline void add(int i, int k)
```

```
{
    for( ; i <= n; i += i & -i) tr[i] += k;
}
// 单点查询
inline int query(int x)
{
    int res = 0;
    for( ; x; x -= x & -x) res += tr[x];
    return res;
}
// 区间查询
inline int query(int x, int y)
{
    int resx = 0, resy = 0;
    for( ; y; y -= y & -y) resy += tr[y];
    for(--x; x; x -= x & -x) resx += tr[x];
    return resy - resx;
}
```

矩阵

矩阵乘法

```
struct Match
{
    long long m[Maxn][Maxn];
    Match(){ memset(m, 0, sizeof m); }
    inline void init() { for(int i = 0; i < Maxn; ++i) m[i][i] = 1; }
    Match operator * (const Match &b) const
    {
        Match res;
        for(int i = 1; i <= n; ++i)
            for(int j = 1; j <= n; ++j)
            {
                long long &cur = res.m[i][j];
                for(int k = 1; k <= n; ++k)
                    cur = (cur+m[i][k]*b.m[k][j])%MOD;
            }
        return res;
    }
}
```

矩阵快速幂 (略)

快速幂

```
inline long long qpow(long long a, long long p, long long mo)
{
    if(p == 0) return 1 % mo;
    long long ans = 1;
    a %= mo;
    while(p)
    {
        if(p&1) ans = ans*a%mo;
        a = a*a%mo;
        p >>= 1;
    }
    return ans;
}
```

最小生成树

Prim

```
inline void prim()
{
    fill(dis, dis+n+1, INF);
    dis[1] = 0;
    for(int t = 1; t <= n; ++t)
    {
        int mini = 0;
        for(int i = 1; i <= n; ++i)
            if(!vis[i] && dis[i] < dis[mini])
                mini = i;
        vis[mini] = 1;
        ans += dis[mini];
        for(int i = 1; i <= n; ++i)
            if(!vis[i]) dis[i] = min(dis[i], calc(mini, i));
    }
}
```

Kruskra (略)

二分图匹配

匈牙利算法

```
// vector<int> _e[Maxn];
bool check(int cur)
{
    for(unsigned i = 0, len = _e[cur].size(); i < len; ++i)
    {
```

```
        int nex = _e[cur][i];
        if(vis[nex]) continue;
        vis[nex] = true;
        if(!co[nex] || check(co[nex]))
        {
            co[nex] = cur;
            return true;
        }
    }
    return false;
}

int solve()
{
    int ans = 0;
    for(int i = 1; i <= n; ++i)
    {
        memset(vis, 0, sizeof vis);
        if(check(i)) ++ans;
    }
    return ans;
}
```

LCA

```
void build_tree(int cur, int fa)
{
    d[cur] = d[fa]+1;
    f[cur][0] = fa;
    for(int i = 1; (1<<i) <= d[cur]; ++i)
        f[cur][i] = f[f[cur][i-1]][i-1];
    for(int k = fir[cur]; k; k = nex[k])
    {
        if(e[k] == fa) continue;
        build_tree(e[k], cur);
    }
}

int lca(int x, int y)
{
    if(d[x] < d[y]) swap(x, y);
    while(d[x] > d[y])
        x = f[x][ lg2[ d[x]-d[y] ]-1 ];
    if(x == y) return x;
    for(int i = lg2[d[x]]; i >= 0; --i)
        if(f[x][i] != f[y][i])
        {
            x = f[x][i];
```

```
            y = f[y][i];
        }
    return f[x][0];
}
```

网络流

增广路

```
// while(bfs()) update();
bool bfs()
{
    memset(v, 0, sizeof v);
    queue<int> q;
    q.push(s);
    v[s] = true;
    incf[s] = INF;
    while(q.size())
    {
        int cur = q.front();
        q.pop();
        for(int i = fir[cur], to; i; i = nex[i])
        {
            to = ver[i];
            if(!edge[i] || v[to]) continue;
            incf[to] = min(incf[cur], edge[i]);
            pre[to] = i;
            if(to == t) return true;
            q.push(to);
            v[to] = true;
        }
    }
    return false;
}

void update()
{
    int cur = t, e;
    while(cur != s)
    {
        e = pre[cur];
        edge[e] -= incf[t];
        edge[e^1] += incf[t];
        cur = ver[e^1];
    }
    maxflow += incf[t];
}
```

Dinic

```
main()
{
    int flow = 0;
    while(bfs())
        while((flow = dinic(s, INF)))
            maxflow += flow;
}

bool bfs()
{
    memset(d, 0, sizeof d);
    queue<int> q;
    q.push(s);
    d[s] = 1;
    int cur;
    while(q.size())
    {
        cur = q.front(); q.pop();
        for(int i = fir[cur], to; i; i = nex[i])
        {
            to = ver[i];
            if(d[to] || !edge[i]) continue;
            d[to] = d[cur]+1;
            if(to == t) return true;
            q.push(to);
        }
    }
    return false;
}

int dinic(int cur, int flow)
{
    if(cur == t) return flow;
    int rest = flow;
    for(int i = fir[cur], to, now; i; i = nex[i])
    {
        to = ver[i];
        if(d[to] != d[cur]+1 || !edge[i]) continue;
        now = dinic(to, min(rest, edge[i]));
        if(!now) d[to] = 0;
        else
        {
            edge[i] -= now;
            edge[i^1] += now;
            rest -= now;
        }
    }
    return flow - rest;
```

```
}
```

最短路

弱化 标准

Floyd

略

Dijiskra

邻接表+堆优化

```
inline void Dijiska()
{
    priority_queue<pair<int,int>,vector<pair<int,int> >,greater<pair<int,int>
> >q;
    memset(dis, 0x7f, sizeof dis);
    dis[S] = 0;
    q.push(make_pair(0, S));
    pair<int, int> cur;
    while(q.size())
    {
        cur = q.top(); q.pop();
        if(dis[cur.second] < cur.first) continue;
        for(int i = fir[cur.second], to, now; i; i = nex[i])
        {
            to = ver[i];
            now = cur.first+w[i];
            if(now >= dis[to]) continue;
            dis[to] = now;
            q.push(make_pair(now, to));
        }
    }
}
```

SPFA

```
inline void SPFA()
{
    fill(dis+1, dis+n+1, INT_MAX);
    dis[S] = 0;
    head = tail = 0;
    q[++tail] = S;
    while(head < tail)
    {
        int cur = q[++head];
```

```
        for(int i = fir[cur], to, tmp; i; i = nex[i])
        {
            to = ver[i];
            tmp = dis[cur]+w[i];
            if(tmp >= dis[to]) continue;
            dis[to] = tmp;
            q[++tail] = to;
        }
    }
}
```

负环

```
// 返回true有负环,返回false没负环
inline bool SPFA()
{
    q[++tail] = 1;
    vis[1] = 1;
    cnt[1] = 1;
    dis[1] = 0;

    while(head < tail)
    {
        int cur = q[(++head)%Maxn];
        vis[cur] = 0;
        for(int i = fir[cur], to; i; i = nex[i])
        {
            to = ver[i];
            if(dis[cur]+w[i] < dis[to])
            {
                dis[to] = dis[cur]+w[i];
                if(!vis[to])
                {
                    q[(++tail)%Maxn] = to;
                    vis[to] = 1;
                    if(++cnt[to] > n) return true;
                }
            }
        }
    }
    return false;
}
```

ST表

```
// for(int i = 2; i <= n; ++i) log_2[i] = log_2[i>>1]+1;
```

```
// st[i][j] --> [i-2^j+1, i]
inline void init_ST(int *a, int st[][20])
{
    for(int i = 1; i <= n; ++i)
    {
        st[i][0] = a[i];
        for(int j = 1; j <= log_2[i]; ++j)
            st[i][j] = max(st[i][j-1], st[i-(1<<(j-1))][j-1]);
    }
}

inline int query(int l, int r, int st[][20])
{
    int k = log_2[r-l+1];
    return max(st[r][k], st[l+(1<<k)-1][k]);
}

// st[i][j] --> [i, i+2^j-1]
void init()
{
    for(int j = 1; j <= log_2[n]; ++j)
        for(int i = 1; 1<<j <= n-i+1; ++i) // i + 1<<j - 1 <= n
            st[i][j] = max(st[i][j-1], st[i+(1<<(j-1))][j-1]);
}

int query(int l, int r)
{
    int k = log_2[r-l+1];
    return max(st[l][k], st[r-(1<<k)+1][k]);
}
```

割点

```
void tarjan(int cur, int fa)
{
    dfn[cur] = low[cur] = ++_dfn;
    int child = 0;
    for(auto i : e[cur])
    {
        if(!dfn[i])
        {
            child++;
            tarjan(i, fa);
            low[cur] = min(low[cur], low[i]);
            if(cur != fa && low[i] >= dfn[cur]) flag[cur] = 1;
        }
        low[cur] = min(low[cur], dfn[i]);
```

```
    }
    if(cur == fa && child >= 2) flag[cur] = 1;
}
```

缩点

```
void tarjan(int cur)
{
    dfn[cur] = low[cur] = ++_dfn;
    q[++tail] = cur;
    vis[cur] = 1;

    for(int i = fir[cur], to; i; i = nex[i])
    {
        to = ver[i];
        if(!dfn[to])
        {
            tarjan(to);
            low[cur] = min(low[cur], low[to]);
        }
        else if(vis[to])
            low[cur] = min(low[cur], dfn[to]);
    }

    if(dfn[cur] == low[cur])
    {
        int sum = 0;
        _col++;
        do
        {
            col[q[tail]] = _col;
            vis[q[tail]] = 0;
            sum += a[q[tail]];
        } while(q[tail--] != cur);
        w[_col] = sum;
    }
}

inline void DAGdp()
{
    for(int i = 1; i <= n; ++i)
        for(int k = fir[i], to; k; k = nex[k])
        {
            to = ver[k];
            if(col[i] != col[to])
            {
                du[col[to]]++;
                add_cedge(col[i], col[to]);
```

```
            }
        }

    head = tail = 0;
    for(int i = 1; i <= _col; ++i)
    {
        if(!du[i])
        {
            q[++tail] = i;
            f[i] = w[i];
            ans = max(ans, f[i]);
        }
    }
    while(head < tail)
    {
        int cur = q[++head];
        for(int k = cfir[cur], to; k; k = cnex[k])
        {
            to = cver[k];
            if(--du[to] == 0)
            {
                q[++tail] = to;
                f[to] = max(f[to], f[cur]+w[to]);
                ans = max(ans, f[to]);
            }
        }
    }
}
```

并查集

```
inline void init() { for(int i = 1; i <= n; ++i) fa[i] = i; }
int getf(int s) { return fa[s] == s ? s : fa[s] = getf(fa[s]); }
inline void connect(int x, int y)
{
    int fx = getf(x), fy = getf(y);
    if(fx != fy) fa[fx] = fy;
}
```

二分

```
int l, r, mid;
while(l < r)
{
    mid = (l+r+1)>>1;
    if(check(mid)) l = mid;
    r = mid-1;
}
while(l < r)
{
    mid = (l+r)>>1;
    if(check(mid)) l = mid+1;
    r = mid;
}
```

欧几里得

最大公因数 gcd

```
int gcd(int a, int b) { return b ? gcd(b, a%b) : a; }
```

最小公倍数 lcm

```
inline int lcm(int a, int b) { return a*b/gcd(a, b); }
```

拓展欧几里得(同余方程)

```
void exgcd(int a, int b, int &x, int &y)
{
    if(!b) { x = 1; y = 0; return; }
    exgcd(b, a%b, y, x);
    y -= a/b*y;
}
```

乘法逆元

拓展欧几里得

```
inline void mul_inverse(int a, int mo)
{
    int x, y;
    exgcd(a, mo, x, y);
    return (x%mo+mo)%mo;
}
```

费马小定理

```
inline void mul_inverse(int a, int mo)
{
    // a^(mo-2)%mo
    return qpow(a, mo-2, mo);
}
```

线性递推

```
inline void mul_inverse(int *inv, int mo)
{
    inv[0] = 0; inv[1] = 1;
    for(int i = 2; i <= n; ++i)
        inv[i] = mo-mo/i*inv[mo%i]%mo;
}
```

线性筛

```
inline void init()
{
    check[1] = true;
    for(int i = 2; i <= n; ++i)
    {
        if(!check[i]) prime[++cnt] = i;
        for(int j = 1; j <= cnt && i*prime[j] <= n; ++j)
        {
            check[ i*prime[j] ] = true;
            if(i % prime[j] == 0) break;
        }
    }
}
```

分解质因数

```
// x = pi^ki...
for(int i = 2; i*i <= x; ++i)
    if(x%i == 0) {
        p[++tot] = i;
        for(; x%i == 0; x /= i) k[tot]++;
    }
if(x > 1) p[++tot] = x, k[tot] = 1;
```

BSGS

luogu 4884

```
// map<long long, int> mmp;
inline long long BSGS(long long a, long long x, long long m) // a^n = x
{
    long long t = (long long)ceil(sqrt(m)); // b = a^i
    for(int i = 0; i < t; ++i)
        mmp[mul(x, qpow(a, i))] = i;
    a = qpow(a, t);
    long long now, ans; // now = (a^t)^i
    for(int i = 0; i <= t; ++i)
    {
        now = qpow(a, i);
        if(mmp.count(now))
        {
            ans = t*i-mmp[now];
            if(ans > 0) return ans;
        }
    }
    return -1;
}
```

背包问题

01背包

完全背包

混合背包

分组背包

多重背包

二进制拆分

```
for(int i = 1, cnt, vi, wi, m; i <= n; ++i)
{
    scanf("%d%d%d", &vi, &wi, &m);
    cnt = 1;
    while(m-cnt > 0)
    {
        m -= cnt;
        v.push_back(vi*cnt);
        w.push_back(wi*cnt);
        cnt <<= 1;
```

```
    }
    v.push_back(vi*m);
    w.push_back(wi*m);
}
for(int i = 0; i < w.size(); ++i)
    for(int j = W; j >= w[i]; --j)
        b[j] = max(b[j], b[j-w[i]]+v[i]);
```

单调队列

```
for(int i = 1; i <= n; ++i)
{
    scanf("%d%d%d", &v, &w, &m);
    for(int u = 0; u < w; ++u)
    {
        int maxp = (W-u)/w;
        head = 1; tail = 0;
        for(int k = maxp-1; k >= max(0, maxp-m); --k)
        {
            while(head <= tail && calc(u, q[tail]) <= calc(u, k)) tail--;
            q[++tail] = k;
        }
        for(int p = maxp; p >= 0; --p)
        {
            while(head <= tail && q[head] >= p) head++;
            if(head <= tail) f[u+p*w] = max(f[u+p*w], p*v+calc(u, q[head]));
            if(p-m-1 < 0) continue;
            while(head <= tail && calc(u, q[tail]) <= calc(u, p-m-1)) tail--;
            q[++tail] = p-m-1;
        }
    }
}
int ans = 0;
for(int i = 1; i <= W; ++i)
    ans = max(ans, f[i]);
```

DP

(我全都不会)

记忆化搜索

线性DP

最长上升子序列LIS

```
for(int i = 1; i <= n; ++i)
{
    f[i] = 1;
    for(int j = 1; j < i; ++j)
        if(a[i] > a[j]) f[i] = max(f[i],f[j]+1);
}
```

最长公共子序列LCS

```
f[i][j] = max{  f[i-1][j],
                f[i][j-1],
                f[i-1][j-1]+1 (if A[i] == B[j])}
```

数字三角形

区间DP

树形DP

状压DP

队列优化

斜率优化

STL

数据结构

```
// const N
// typename T
vector<T>
stack<T>
deque<T>
queue<T>
priority_queue<T>
set<T>
bitset<N>
map<T, T>
```

函数

```
// vector<T> a;
sort(a.begin(), a.end(), cmp);
reverse(a.begin(), a.end());
unique(a.begin(), a.end());
```

```
next_permutation(a.begin(), a.end());
lower_bound(a.begin(), a.end(), val); // >=
upper_bound(a.begin(), a.end(), val); // >
fill(a.begin(), a.end(), val);
memset(a, 0, sizeof a);
memcpy(a, b, sizeof b); // b --> a
// string s
s.find(target_string, start_pos); // 找不到返回s.npos
s.substr(start_pos, len);
s.replace(start_pos, len, target_string);
```

高精度

压位+vector+符号 版本 一本通习题 洛谷习题

此版本 压位+数组,支持cin,cout,string,long long转换,比较运算符,四则运算(包括高精度乘/除低精度,取模),支持带符号的减法运算,支持幂运算,开根运算

可以通过开根外所有习题

```
struct BigInteger
{
    static const int SIZE = 1e4; // 位数SIZE*4
    static const int BASE = 1e4; // 压位
    static const int WIDTH = 4;

    int v[SIZE], len;
    int tag; // 假装有正负符号

    BigInteger(long long num = 0) { *this = num; }
    BigInteger(const string &str) { *this = str; }
    // long long 转 BigInteger
    BigInteger operator = (long long num)
    {
        len = tag = 0;
        memset(v, 0, sizeof v);
        do
        {
            v[++len] = (int)(num%BASE);
            num /= BASE;
        } while(num > 0);
        return *this;
    }
    // string 转 BigInteger
    BigInteger operator = (const string &str)
    {
        string buf;
        int r = (int)str.length()-1, l = max(0, r-WIDTH+1);
        len = tag = 0;
```

```
        memset(v, 0, sizeof v);
        while(r >= 0)
        {
            buf = str.substr(l, r-l+1);
            sscanf(buf.c_str(), "%d", &v[++len]);
            r -= WIDTH; l = max(0, r-WIDTH+1);
        }
        return *this;
    }
    // 比较运算
    bool operator < (const BigInteger &b) const
    {
        if(len != b.len) return len < b.len;
        for(int i = len; i; --i)
            if(v[i] != b.v[i]) return v[i] < b.v[i];
        return false;
    }

    bool operator > (const BigInteger &b) const { return b < *this; }
    bool operator <= (const BigInteger &b) const { return !(b < *this); }
    bool operator >= (const BigInteger &b) const { return !(*this < b); }
    bool operator != (const BigInteger &b) const { return *this < b || b <
*this; }
    bool operator == (const BigInteger &b) const { return !(*this < b) && !(b
< *this); }
    // 四则运算
    BigInteger operator + (const BigInteger &b) const
    {
        BigInteger res = b;
        res.len = max(len, b.len);
        for(int i = 1; i <= len; ++i)
            res.v[i] += v[i];
        for(int i = 1; i <= res.len; ++i)
            res.v[i+1] += res.v[i]/BASE,
            res.v[i] %= BASE;
        while(res.v[res.len+1] > 0) res.len++;
        return res;
    }
    // 单目运算
    BigInteger operator + () const { return *this; }
    BigInteger operator - () const
    {
        BigInteger res = *this;
        res.tag ^= 1;
        return res;
    }

    BigInteger operator - (const BigInteger &b) const
    {
        if(*this < b) return -(b-*this);
        BigInteger res = *this;
```

```
    for(int i = 1; i <= b.len; ++i)
        res.v[i] -= b.v[i];
    for(int i = 1; i <= res.len; ++i)
        if(res.v[i] < 0)
            res.v[i] += BASE,
            res.v[i+1]--;
    while(res.len > 1 && res.v[res.len] == 0) res.len--;
    return res;
}
// 高精度乘低精度
BigInteger operator * (int b) const
{
    BigInteger res;
    long long tmp;
    res.len = len;
    for(int i = 1; i <= len; ++i)
    {
        tmp = 1ll*b*v[i];
        res.v[i] += (int)(tmp%BASE);
        res.v[i+1] += (int)(tmp/BASE+res.v[i]/BASE);
        res.v[i] %= BASE;
    }
    while(res.v[res.len+1] > 0) res.len++;
    return res;
}
// 高精度乘高精度
BigInteger operator * (const BigInteger &b) const
{
    BigInteger res;
    res.len = len+b.len;
    for(int i = 1; i <= len; ++i)
        for(int j = 1; j <= b.len; ++j)
        {
            res.v[i+j-1] += v[i]*b.v[j];
            res.v[i+j] += res.v[i+j-1]/BASE;
            res.v[i+j-1] %= BASE;
        }
    while(res.len > 1 && res.v[res.len] == 0) res.len--;
    return res;
}
// 高精度除低精度
BigInteger operator / (int b) const
{
    long long divisor = 0;
    BigInteger res;
    for(int i = len; i; --i)
    {
        divisor = divisor*BASE+v[i];
        if(divisor < b) continue;
        res.v[i] = (int)(divisor/b);
        divisor %= b;
```

```
            res.len = max(res.len, i);
        }
        return res;
    }
    // 高精度除高精度
    BigInteger operator / (const BigInteger &b) const
    {
        BigInteger divisor, res;
        int l, r, mid;
        for(int i = len; i; --i)
        {
            divisor = divisor*BASE+v[i];
            /*
            memcpy(divisor.v+1, divisor.v, sizeof(int)*(divisor.len+1));
            while(divisor.v[divisor.len+1] > 0) divisor.len++;
            divisor.v[1] = v[i];
            */
            if(divisor < b) continue;
            l = 0; r = BASE-1;
            while(l < r)
            {
                mid = (l+r+1)>>1;
                if(b*mid <= divisor) l = mid;
                else r = mid-1;
            }
            divisor -= b*l;
            res.v[i] = l;
            res.len = max(res.len, i);
        }
        return res;
    }

    BigInteger operator % (const BigInteger &b) const { return *this-
*this/b*b; }

    BigInteger operator ++ () { return *this = *this+1; }
    BigInteger operator -- () { return *this = *this-1; }
    BigInteger operator += (const BigInteger &b) { return *this = *this+b; }
    BigInteger operator -= (const BigInteger &b) { return *this = *this-b; }
    BigInteger operator *= (const BigInteger &b) { return *this = *this*b; }
    BigInteger operator /= (const BigInteger &b) { return *this = *this/b; }
    BigInteger operator %= (const BigInteger &b) { return *this = *this%b; }
    BigInteger operator *= (int b) { return *this = *this*b; }
    BigInteger operator /= (int b) { return *this = *this/b; }
    BigInteger operator %= (int b) { return *this = *this%b; }
};

// 重载输入运算符
istream &operator >> (istream &in, BigInteger &big)
{
    string buf;
```

```
    if(in >> buf) big = buf;
    return in;
}
// 重载输出运算符
ostream &operator << (ostream &os, const BigInteger &big)
{
    char buf[10];
    if(big.tag) os << '-';
    os << big.v[big.len];
    for(int i = big.len-1; i; --i)
    {
        sprintf(buf, "%04d", big.v[i]);
        for(int j = 0; j < 4; ++j) os << buf[j];
    }
    return os;
}
// 幂
template <typename T>
BigInteger pow (BigInteger a, T p)
{
    if(p == 0) return 1;
    BigInteger res = 1;
    while(p)
    {
        if(p%2) res *= a;
        a *= a;
        p /= 2;
    }
    return res;
}
// 开根
BigInteger sqrt(const BigInteger &a, const int p = 2)
{
    BigInteger l, r = a, mid;
    while(l < r)
    {
        mid = (l+r+1)/2;
        if(pow(mid, p) <= a) l = mid;
        else r = mid-1;
    }
    return l;
}
```