数据结构

ST表

> f[i][j]:从第 i 个数开始区间长度为 2^j 的区间最大值

```
void init(int n)
{
    for(int j = 0; j < M; j ++)
        for(int i = 1; i + (1 << j) - 1 <= n; i ++)
            if(!j)   f[i][j] = w[i];
            else   f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
}

int query(int l, int r)
{
    int len = r - l + 1;
    int k = log(len) / log(2);

    return max(f[l][k], f[r - (1 << k) + 1][k]);
}
```

树状数组

```
int lowbit(int x)
{
    return x & -x;
}

void add(int x, int c)
{
    for(int i = x; i < N; i += lowbit(i))
        tr[i] += c;
}

LL query(int x)
```

```
{
    LL res = 0;
    for(int i = x; i; i -= lowbit(i))
        res += tr[i];
    return res;
}
```

线段树

```
struct Node
{
    int l, r;
    LL sum, add;
}tr[N << 2];

void pushup(int u)
{
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

void pushdown(int u)
{
    Node& root = tr[u], & l = tr[u << 1], & r = tr[u << 1 | 1];
    if(root.add)
    {
        l.add += root.add, l.sum += (LL)(l.r - l.l + 1) * root.add;
        r.add += root.add, r.sum += (LL)(r.r - r.l + 1) * root.add;
        root.add = 0;
    }
}

void build(int u, int l, int r)
{
    tr[u] = {l, r};
    if(l == r)
        tr[u].sum = w[l];
    else
    {
        int mid = l + r >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, int d)
{
    if(l <= tr[u].l && tr[u].r <= r)
        tr[u].sum += (LL)(tr[u].r - tr[u].l + 1) * d, tr[u].add += d;
    else
    {
        pushdown(u);

        int mid = tr[u].l + tr[u].r >> 1;
        if(l <= mid)    modify(u << 1, l, r, d);
        if(r > mid)    modify(u << 1 | 1, l, r, d);

        pushup(u);
    }
}

LL query(int u, int l, int r)
{
    if(l <= tr[u].l && tr[u].r <= r)
        return tr[u].sum;

    pushdown(u);

    int mid = tr[u].l + tr[u].r >> 1;
    LL res = 0;
    if(l <= mid)    res += query(u << 1, l, r);
    if(r > mid)    res += query(u << 1 | 1, l, r);

    return res;
}
```

动态规划

换根dp

```
void add(int a, int b, int c)
{
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

void link(int u, int v, int w)
{
    sz[u] += sz[v];
    if(sz[v])
    {
        f[u] += f[v] + w;

        LL t = g[v][0] + w;
        if(t >= g[u][0])
            g[u][1] = g[u][0], g[u][0] = t;
        else if(t >= g[u][1])
            g[u][1] = t;
    }
}

void cut(int u, int v, int w)
{
    sz[u] -= sz[v];
    if(sz[v])
    {
        f[u] -= f[v] + w;

        LL t = g[v][0] + w;
        if(t == g[u][0])
```

```
            g[u][0] = g[u][1], g[u][1] = 0;
        else if(t == g[u][1])
            g[u][1] = 0;
    }
}

auto dfs1 = [&](auto dfs1, int u, int fa) -> void
{
    for(int i = h[u]; ~i; i = ne[i])
    {
        int v = e[i];
        if(v == fa)     continue;

        dfs1(dfs1, v, u);
        link(u, v, w[i]);
    }
};

auto dfs2 = [&](auto dfs2, int u, int fa) -> void
{
    ans[u] = f[u] * 2 - g[u][0];

    for(int i = h[u]; ~i; i = ne[i])
    {
        int v = e[i];
        if(v == fa)     continue;

        cut(u, v, w[i]);
        link(v, u, w[i]);

        dfs2(dfs2, v, u);

        cut(v, u, w[i]);
        link(u, v, w[i]);
    }
};
```

字符串

字符串哈希

初始化

```
void init(int n)
{
    p[0] = 1;
    for(int i = 1; i <= n; i ++)
        p[i] = p[i - 1] * P, h[i] = h[i - 1] * P + s[i];
}
```

访问

```
ui128 get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}
```

字符串最小表示法

```
for(int i = 1; i <= n; i ++)
    s[i + n] = s[i];

int I = 1, J = 2;
while(J <= n)
{
    for(int k = 0; k < n; k ++)
        if(s[I + k] < s[J + k])
        {
            J += k;
            break;
        }
        else if(s[I + k] > s[J + k])
        {
            int t = I;
            I = J, J = max(J, t + k);
            break;
        }

    J ++;
}
```