icpc模板

模板

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;
typedef pair<int, int> PII;

const int N = 1e6 + 10;
const LL mod = 998244353;

void Solve()
{

}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T=1;
    cout << fixed << setprecision(10);
    cin >> T; while(T--)
    Solve();
    return 0;
}
```

并查集

```
struct DSU
{
    std::vector<int> p, siz;

    DSU() {}
```

```
    DSU(int n) { init(n); }

    int find(int x)
    {
        if (x != p[x]) p[x] = find(p[x]);
        return p[x];
    }
    bool same(int x, int y) { return find(x) == find(y); }
    bool merge(int x, int y)
    {
        x = find(x);
        y = find(y);
        if (x == y) return false;

        p[y] = x;
        siz[x] += siz[y];
        return true;
    }
    int size(int x) { return siz[find(x)]; }

private:
    void init(int n)
    {
        p.resize(n + 1);
        std::iota(p.begin(), p.end(), 0);
        siz.assign(n + 1, 1);
    }
};
```

应用1：洛谷P1111

第 1 行两个正整数 N, M。

下面 M 行，每行 3 个正整数 x, y, t，告诉你这条公路连着 x, y 两个村庄，在时间t时能修复完成这条公路。

如果全部公路修复完毕仍然存在两个村庄无法通车，则输出 -1，否则输出最早什么时候任意两个村庄能够通车。

```
void luogu_P1111()
{
    int n, m;
    cin >> n >> m;
```

```
    DSU dsu(n);
    vector<tuple<int, int, int>> a(m);

    for (int i = 0; i < m; i++)
    {
        int x, y, t;
        cin >> x >> y >> t;
        a[i] = make_tuple(t, x, y);
    }

    sort(a.begin(), a.end());

    for (int i = 0; i < m; i++)
    {
        dsu.merge(get<1>(a[i]), get<2>(a[i]));

        if (dsu.size(get<1>(a[i])) == n)
        {
            cout << get<0>(a[i]) << "\n";
            return;
        }
    }

    cout << -1 << "\n";
}
```

应用2：洛谷P1525

公务繁忙的 Z 市长只会去看列表中的第一个事件的影响力，警察局长准备将罪犯们在两座监狱内重新分配，应如何分配罪犯，才能使 Z 市长看到的那个冲突事件的影响力最小？这个最小值是多少?

输入每行中两个数之间用一个空格隔开。第一行为两个正整数 N, M，分别表示罪犯的数目以及存在仇恨的罪犯对数。接下来的 M 行每行为三个正整数 a_j, b_j, c_j，表示 a_j 号和 b_j 号罪犯之间存在仇恨，其怨气值为 c_j。数据保证 $1 < a_j \leq b_j \leq N, 0 < c_j \leq 10^9$，且每对罪犯组合只出现一次。

输出共一行，为 Z 市长看到的那个冲突事件的影响力。如果本年内监狱中未发生任何冲突事件，请输出 `0` 。

```
void luogu_P1525()
{
    int n, m;
```

```
    cin >> n >> m;

    DSU dsu(n);
    vector<array<int, 3>> edges(m);
    for (int i = 0; i < m; i++)
    {
        int a, b, w;    cin >> a >> b >> w;
        edges[i] = { w, a, b };
    }
    sort(edges.begin(), edges.end(), greater<array<int, 3>>());

    // 记录另一个监狱中对应的人
    vector<int> others(n + 1, -1);
    for (int i = 0; i < m; i++)
    {
        int w = edges[i][0], a = edges[i][1], b = edges[i][2];
        if (dsu.same(a, b))
        {
            cout << w << "\n";
            return;
        }
        if (others[a] == -1) others[a] = b;
        else dsu.merge(others[a], b);

        if (others[b] == -1) others[b] = a;
        else dsu.merge(others[b], a);
    }
    cout << 0 << "\n";
}
```

应用3（种类并查集）：洛谷P2024

动物王国中有三类动物 A,B,C，这三类动物的食物链构成了有趣的环形。A 吃 B，B 吃 C，C 吃 A。

现有 N 个动物，以 $1\sim N$ 编号。每个动物都是 A,B,C 中的一种，但是我们并不知道它到底是哪一种。

有人用两种说法对这 N 个动物所构成的食物链关系进行描述：

- 第一种说法是 `1 X Y`，表示 X 和 Y 是同类。
- 第二种说法是 `2 X Y`，表示 X 吃 Y。

此人对 N 个动物，用上述两种说法，一句接一句地说出 K 句话，这 K 句话有的是真的，有的是假的。当一句话满足下列三条之一时，这句话就是假话，否则就是真话。

- 当前的话与前面的某些真的话冲突，就是假话；

- 当前的话中 X 或 Y 比 N 大，就是假话；
- 当前的话表示 X 吃 X，就是假话。

你的任务是根据给定的 N 和 K 句话，输出假话的总数。

```
void luogu_P2024()
{
    int n, k; cin >> n >> k;
    DSU dsu(3 * n);
    int res = 0;
    for (int i = 0; i < k; i++) {
        int op, x, y;   cin >> op >> x >> y;
        if (x > n || y > n) { res++; continue; }
        if (op == 1) {
            if (dsu.same(x, y + n) || dsu.same(x + n, y)) res++;
            else {
                dsu.merge(x, y); dsu.merge(x + n, y + n); dsu.merge(x + n + n,
y + n + n);
            }
        }else {
            if (dsu.same(x, y) || dsu.same(x, y + n)) res++;
            else {
                dsu.merge(x + n, y); dsu.merge(x + n + n, y + n); dsu.merge(x,
y + n + n);
            }
        }
    }
    cout << res << "\n";
}
```

带权并查集

```
struct DSU
{
    std::vector<int> p, siz, val;

    DSU() {}
    DSU(int n) { init(n); }

    int find(int x)
    {
        if (x != p[x])
        {
            int px = find(p[x]);
```

```
            val[x] += val[p[x]];
            p[x] = px;
        }
        return p[x];
    }
    bool same(int x, int y) { return find(x) == find(y); }
    bool merge(int x, int y)
    {
        x = find(x);
        y = find(y);
        if (x == y) return false;

        p[x] = y;
        val[x] += siz[y];
        siz[y] += siz[x];
        return true;
    }
    int size(int x) { return siz[find(x)]; }

private:
    void init(int n)
    {
        p.resize(n + 1);
        std::iota(p.begin(), p.end(), 0);
        siz.assign(n + 1, 1);
        val.assign(n + 1, 0);
    }
};
```

树状数组

```
template <typename T>
struct Fenwick
{
    int n;
    std::vector<T> tr;

    Fenwick(int n_ = 0) { init(n_); }

    void add(int x, const T& v) { for (int i = x; i ≤ n; i += lowbit(i))
tr[i] = tr[i] + v; }
    T query(int x)
```

```
    {
        T ans{};
        for (int i = x; i; i -= lowbit(i)) ans = ans + tr[i];
        return ans;
    }
    // sum[l, r]
    T rangeSum(int l, int r) { return query(r) - query(l - 1); }

private:
    void init(int n_)
    {
        n = n_;
        tr.assign(n + 1, T{});
    }

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

应用1：洛谷P3374

```
void luogu_P3374()
{
    int n, m;
    cin >> n >> m;
    Fenwick<LL> fen(n);

    for (int i = 1; i <= n; i++)
    {
        LL x; cin >> x;
        fen.add(i, x);
    }

    for (int i = 0; i < m; i++)
    {
        int op; cin >> op;
        // 1 x k 含义：将第 x 个数加上k
        if (op == 1)
        {
            int x;
            LL k;
            cin >> x >> k;
            fen.add(x, k);
        }
        // 2 x y 含义：输出区间 [x, y] 内每个数的和
        else
        {
            int x, y;
            cin >> x >> y;
            cout << fen.rangeSum(x, y) << "\n";
        }
    }
}
```

应用2：洛谷P3368

```
void luogu_P3368()
{
    int n, m;
    cin >> n >> m;
    vector<LL> a(n + 1, 0);

    for (int i = 1; i ⩽ n; i++) cin >> a[i];

    Fenwick<LL> fen(n + 1);
    for (int i = 1; i ⩽ n; i++) fen.add(i, a[i] - a[i - 1]);

    for (int i = 0; i < m; i++)
    {
        int op; cin >> op;
        // 1 x y k 含义：将区间 [x, y] 内每个数加上k;
        if (op == 1)
        {
            int x, y;
            LL k;
            cin >> x >> y >> k;
            fen.add(x, k);
            fen.add(y + 1, -k);
        }
        // 2 x 含义：输出第 x 个数的值
        else
        {
            int x;
            cin >> x;
            cout << fen.query(x) << "\n";
        }
    }
}
```

应用3（超级树状数组）：洛谷P3372

$$prefix = \sum_{i=1}^{x} d[i] * (x+1) - d[i] * i$$

```
void luogu_P3372()
{
    int n, m;
    cin >> n >> m;

    vector<LL> a(n + 1);
    for (int i = 1; i ⩽ n; i++) cin >> a[i];

    Fenwick<LL> f1(n + 1), f2(n + 1);

    auto add = [&](int x, LL v) {f1.add(x, v); f2.add(x, v * x); };
    auto query = [&](int x) {return f1.query(x) * (x + 1) - f2.query(x); };

    for (int i = 1; i ⩽ n; i++)
    {
        add(i, a[i]); add(i + 1, -a[i]);
    }

    for (int i = 0; i < m; i++)
    {
        int op; cin >> op;
        // 1 x y k: 将区间 [x, y] 内每个数加上k。
        if (op == 1)
        {
            int x, y;
            LL k;
            cin >> x >> y >> k;
            add(x, k); add(y + 1, -k);
        }
        // 2 x y: 输出区间 [x, y] 内每个数的和。
        else
        {
            int x, y;
            cin >> x >> y;
            cout << query(y) - query(x - 1) << "\n";
        }
    }
}
```

应用4：洛谷P1972

求区间种类数：离线+树状数组

```
void luogu_P1972()
{
    int n;  cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];

    int m;  cin >> m;
    vector<array<int, 3>> query(m);
    vector<int> ans(m);
    for (int i = 0; i < m; i++)
    {
        int x, y;   cin >> x >> y;
        query[i] = { x, y, i };
    }
    sort(query.begin(), query.end(), [](array<int, 3> x, array<int, 3> y)
        {
            return x[1] < y[1];
        });

    map<int, int> pos;
    Fenwick<int> w(n);
    for (int i = 1, j = 0; i <= n; i++)
    {
        if (pos.count(a[i])) w.add(pos[a[i]], -1);
        pos[a[i]] = i;
        w.add(i, 1);

        while (j < m && query[j][1] == i)
        {
            int l = query[j][0], r = query[j][1], idx = query[j][2];
            ans[idx] = w.rangeSum(l, r);
            j++;
        }
    }

    for (int i = 0; i < m; i++) cout << ans[i] << "\n";
}
```

二维树状数组

```
template <typename T>
struct Fenwick2D
{
    int n, m;
    std::vector<std::vector<T>> tr;

    Fenwick2D(int n_ = 0, int m_ = 0) { init(n_, m_); }
    void add(int x, int y, const T& v)
    {
        for (int i = x; i ⩽ n; i += lowbit(i))
            for (int j = y; j ⩽ m; j += lowbit(j))
                tr[i][j] = tr[i][j] + v;
    }
    T query(int x, int y)
    {
        T ans{};
        for (int i = x; i; i -= lowbit(i))
            for (int j = y; j; j -= lowbit(j))
                ans = ans + tr[i][j];
        return ans;
    }
    T rangeSum(int x1, int y1, int x2, int y2)
    {
        return query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) +
query(x1 - 1, y1 - 1);
    }

private:
    void init(int n_, int m_)
    {
        n = n_;
        m = m_;
        tr.assign(n + 1, std::vector<T>(m + 1, T{}));
    }
    int lowbit(int x) { return x & -x; }
};
```

应用：洛谷P4514

区间修改，区间查询

$$prefix = \sum_{i=1}^{x}\sum_{j=1}^{y} d[i][j] * (xy + x + y + 1) - d[i][j] * i(y + 1) - d[i][j] * j(x + 1) + d[i][j] * i * j$$

```
void luogu_P4514()
{
    string op;
    int n, m;
    cin >> op >> n >> m;
    n++, m++;
    Fenwick2D<int> f(n, m), fi(n, m), fj(n, m), fij(n, m);

    auto add = [&](int x, int y, int v)
    {
        f.add(x, y, v);         fi.add(x, y, v * x);
        fj.add(x, y, v * y);    fij.add(x, y, v * x * y);
    };
    auto query = [&](int x, int y)
    {
        return f.query(x, y) * (x * y + x + y + 1) -
            fi.query(x, y) * (y + 1) -
            fj.query(x, y) * (x + 1) +
            fij.query(x, y);
    };

    while (cin >> op)
    {
        if (op == "L")
        {
            int x1, y1, x2, y2, v;  cin >> x1 >> y1 >> x2 >> y2 >> v;
            add(x1, y1, v);         add(x2 + 1, y1, -v);
            add(x1, y2 + 1, -v);    add(x2 + 1, y2 + 1, v);
        }
        else
        {
            int x1, y1, x2, y2;     cin >> x1 >> y1 >> x2 >> y2;
            cout << query(x2, y2) - query(x1 - 1, y2) - query(x2, y1 - 1) +
query(x1 - 1, y1 - 1) << "\n";
        }
    }
}
```

线段树：基础区间加乘

```
struct SegmentTree
{
    int n;
    std::vector<LL> tag, sum;
    LL mod;

    SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) { mod = 1e18; }
    SegmentTree(int n_, LL mod_) : n(n_), tag(4 * n, 1), sum(4 * n), mod(mod_)
{ }

    // [x, y)
    LL query(int x, int y) { return query(1, 0, n, x, y); }
    void rangeMul(int x, int y, int v) { rangeMul(1, 0, n, x, y, v); }
    void add(int x, int v) { add(1, 0, n, x, v); }

private:
    void pull(int p) { sum[p] = (sum[2 * p] + sum[2 * p + 1]) % mod; }

    void mul(int p, LL v)
    {
        tag[p] = 1LL * tag[p] * v % mod;
        sum[p] = 1LL * sum[p] * v % mod;
    }

    void push(int p)
    {
        mul(2 * p, tag[p]);
        mul(2 * p + 1, tag[p]);
        tag[p] = 1;
    }

    LL query(int p, int l, int r, int x, int y)
    {
        if (l ⩾ y || r ⩽ x) return 0;
        if (l ⩾ x && r ⩽ y) return sum[p];

        int m = (l + r) / 2;
        push(p);
        return (query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y)) %
mod;
    }
```

```
    void rangeMul(int p, int l, int r, int x, int y, int v)
    {
        if (l ≥ y || r ≤ x) return;
        if (l ≥ x && r ≤ y) return mul(p, (LL)v);

        int m = (l + r) / 2;
        push(p);
        rangeMul(2 * p, l, m, x, y, v);
        rangeMul(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void add(int p, int l, int r, int x, int v)
    {
        if (r - l == 1)
        {
            sum[p] = (sum[p] + v) % mod;
            return;
        }

        int m = (l + r) / 2;
        push(p);
        if (x < m) add(2 * p, l, m, x, v);
        else add(2 * p + 1, m, r, x, v);
        pull(p);
    }
};
```

懒标记线段树：基础区间修改

```
template<class Info, class Tag>
struct LazySegmentTree
{
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree(int n) : n(n), info(n << 2), tag(n << 2) {}
    LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size())
    {
        std::function<void(int, int, int)> build = [&](int p, int l, int r)
        {
            if (l == r)
            {
                info[p] = init[l];
```

```
            return;
        }

        int mid = l + r >> 1;
        build(2 * p, l, mid);
        build(2 * p + 1, mid + 1, r);
        push_up(p);
    };
    build(1, 0, n - 1);
}
void push_up(int p)
{
    info[p] = info[2 * p] + info[2 * p + 1];
}
void addtag(int p, const Tag& v)
{
    info[p].apply(v);
    tag[p].apply(v);
}
void push_down(int p)
{
    addtag(2 * p, tag[p]);
    addtag(2 * p + 1, tag[p]);
    tag[p] = Tag();
}
// 单点修改
void modify(int p, int l, int r, int x, const Info& v)
{
    if (l == r)
    {
        info[p] = v;
        return;
    }
    push_down(p);
    int mid = l + r >> 1;
    if (x <= mid) modify(2 * p, l, mid, x, v);
    else modify(2 * p + 1, mid + 1, r, x, v);
    push_up(p);
}
void modify(int p, const Info& v)
{
    modify(1, 0, n - 1, p, v);
}
Info rangeQuery(int p, int l, int r, int x, int y)
{
    if (l > y || r < x) return Info();
    if (l >= x && r <= y) return info[p];
```

```
        push_down(p);
        int mid = l + r >> 1;
        return rangeQuery(2 * p, l, mid, x, y) + rangeQuery(2 * p + 1, mid +
1, r, x, y);
    }
    Info rangeQuery(int l, int r)
    {
        return rangeQuery(1, 0, n - 1, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag& v)
    {
        if (l > y || r < x) return;
        if (l ⩾ x && r ⩽ y)
        {
            addtag(p, v);
            return;
        }
        push_down(p);
        int mid = l + r >> 1;
        rangeApply(2 * p, l, mid, x, y, v);
        rangeApply(2 * p + 1, mid + 1, r, x, y, v);
        push_up(p);
    }
    void rangeApply(int l, int r, const Tag& v)
    {
        rangeApply(1, 0, n - 1, l, r, v);
    }
};

typedef long long i64;
constexpr i64 inf = 1e18;

struct Tag
{
    i64 add = 0;

    void apply(Tag t)
    {
        add += t.add;
    }
};

struct Info
{
    i64 minv = inf;
    i64 maxv = -inf;
    i64 sum = 0;
```

```
    i64 act = 0;

    void apply(Tag t)
    {
        minv += t.add;
        maxv += t.add;
        sum += act * t.add;
    }
};

Info operator+(Info a, Info b)
{
    Info c;
    c.minv = std::min(a.minv, b.minv);
    c.maxv = std::max(a.maxv, b.maxv);
    c.sum = a.sum + b.sum;
    c.act = a.act + b.act;
    return c;
}
```

FHQ-Treap

```
struct Treap
{
    struct node
    {
        int ls, rs;
        int key, val;
        int size;
        void init(int v)
        {
            ls = rs = 0;
            size = 1;
            key = rand();
            val = v;
        }
    };

    vector<node> tr;
    int cnt;
```

```
    int root;

    Treap(int n) {init(n);}
    void newnode(int x)
    {
        cnt++;
        tr[cnt].init(x);
    }
    void push_up(int u)
    {
        tr[u].size = tr[tr[u].ls].size + tr[tr[u].rs].size + 1;
    }
    void split(int u, int v, int &x, int &y)
    {
        if(u == 0)
        {
            x = y = 0;
            return;
        }
        if(tr[u].val > v)
        {
            y = u;
            split(tr[u].ls, v, x, tr[u].ls);
        }
        else
        {
            x = u;
            split(tr[u].rs, v, tr[u].rs, y);
        }
        push_up(u);
    }
    int merge(int x, int y)
    {
        if(x == 0 || y == 0) return x + y;
        if(tr[x].key > tr[y].key)
        {
            tr[x].rs = merge(tr[x].rs, y);
            push_up(x);
            return x;
        }
        else
        {
            tr[y].ls = merge(x, tr[y].ls);
            push_up(y);
            return y;
        }
    }
```

```
    void init(int n)
    {
        tr.resize(n+1);
        cnt = 0;
        root = 0;
    }
};
```

杂项

离散化

```
vector<int> b(a);
sort(b.begin(), b.end());
b.erase(unique(b.begin(), b.end()), b.end());

// 查找a中某个数离散化后的结果
auto getval = [&](int x) {return lower_bound(b.begin(), b.end(), x) -
b.begin() + 1; };
```

Mint类

```
template<int mod>
class mint
{
public:
    unsigned int x = 0;
    mint inv() const { return pow(mod - 2); }
    mint pow(long long t) const {
        assert(t ⩾ 0 && x > 0);
        mint res = 1, cur = x;
        for (; t; t ⩾= 1) {
            if (t & 1) res *= cur;
            cur *= cur;
        }
        return res;
    }
    mint() = default;
    mint(unsigned int t) :x(t% mod) { }
    mint(int t) {
        t %= mod;
        if (t < 0) t += mod;
        x = t;
    }
    mint(long long t) {
        t %= mod;
        if (t < 0) t += mod;
```

```
        x = t;
    }

    mint& operator+=(const mint& t) {
        x += t.x;
        if (x ⩾ mod) x -= mod;
        return *this;
    }
    mint& operator-=(const mint& t) {
        x += mod - t.x;
        if (x ⩾ mod) x -= mod;
        return *this;
    }
    mint& operator*=(const mint& t) {
        x = (unsigned long long)x * t.x % mod;
        return *this;
    }
    mint& operator/=(const mint& t) {
        *this *= t.inv();
        return *this;
    }
    mint& operator^=(const mint& t) {
        *this = this->pow(t.x);
        return *this;
    }

    mint operator+(const mint& t) { return mint(*this) += t; }
    mint operator-(const mint& t) { return mint(*this) -= t; }
    mint operator*(const mint& t) { return mint(*this) *= t; }
    mint operator/(const mint& t) { return mint(*this) /= t; }
    mint operator^(const mint& t) { return mint(*this) ^= t; }

    bool operator==(const mint& t) { return x == t.x; }
    bool operator≠(const mint& t) { return x ≠ t.x; }
    bool operator<(const mint& t) { return x < t.x; }
    bool operator⩽(const mint& t) { return x ⩽ t.x; }
    bool operator>(const mint& t) { return x > t.x; }
    bool operator⩾(const mint& t) { return x ⩾ t.x; }

    friend istream& operator>>(istream& is, mint& t) { return is >> t.x; }
    friend ostream& operator<<(ostream& os, const mint& t) { return os << t.x;
}
    friend mint operator+(int y, const mint& t) { return mint(y) + t.x; }
    friend mint operator-(int y, const mint& t) { return mint(y) - t.x; }
    friend mint operator*(int y, const mint& t) { return mint(y) * t.x; }
    friend mint operator/(int y, const mint& t) { return mint(y) / t.x; }
};
```

组合数

```
LL f[N], g[N], inv[N];

void init()
{
    f[0] = g[0] = inv[0] = 1;
    for(int i=1; i<N; i++)
    {
        if(i == 1) inv[i] = 1;
        else inv[i] = (mod - mod / i) * inv[mod % i] % mod;
        f[i] = f[i-1] * i % mod;
        g[i] = g[i-1] * inv[i] % mod;
    }

}
```

字符串

字符串哈希

单哈希(BKDRHash)

```
typedef unsigned long long ull;

// s: 1 - index
ull BKDRHash(string s)
{
    ull P = 131, H = 0;
    int n = s.size() - 1;
    for(int i=1; i<=n; i++)
        H = H * P + s[i];
    return H;
}
```

封装后的写法（容易被卡）

```
typedef unsigned long long ull;
struct strHash
{
    static const ull base = 131;
    int n;
    vector<ull> H;
    vector<ull> P;
    // s: 1 - index
    strHash(string s): n(s.size()-1), H(n+1), P(n+1)
    {
        P[0] = 1;
        for(int i=1; i<=n; i++)
        {
            P[i] = P[i-1] * base;
            H[i] = H[i-1] * base + s[i];
        }
    }
    ull get_hash(int l, int r)
    {
        return H[r] - H[l-1] * P[r-l+1];
    }
};
```

Manacher算法

```
string change(string s)
{
    string t = "$#";
    for(auto c: s)
    {
        t += c;
        t += '#';
    }
    t += '&';
    return t;
}
vector<int> manacher(string s)
{
    string t = change(s);
    int n = t.size();
```

```
    vector<int> P(n);
    int R = 0, C;

    for(int i=1; i<n-1; i++)
    {
        if(i < R) P[i] = min(P[(C<<1)-i], P[C]+C-i);
        else P[i] = 1;

        while(t[i-P[i]] == t[i+P[i]]) P[i]++;
        if(i+P[i] > R)
        {
            R = P[i] + i;
            C = i;
        }
    }

    return P;
}
```

Z函数

```
vector<int> zFunction(string s)
{
    int n = s.size();
    vector<int> z(n+1);
    z[0] = n;
    for(int i=1, j=1; i<n; i++)
    {
        z[i] = max(0, min(j + z[j] - i, z[i - j]));
        while(i + z[i] < n && s[z[i]] == s[i + z[i]])
            z[i]++;
        if(i + z[i] > j + z[j])
            j = i;
    }
    return z;
}
```

字典树

洛谷P2580

```
struct node
{
    bool repeat;    // 这个前缀是否重复
    int son[26];    // 26个字母
    int num;        // 这个前缀出现的次数
    bool isend;     // 是否为单词的结尾
};

struct Trie
{
    vector<node> tr;
    int cnt;
    Trie(int n, int maxsize = 1)
    {
        tr.resize(n * maxsize + 5);
        cnt = 1;
    }
    void insert(string str)
    {
        int now = 0;
        for(auto c: str)
        {
            int ch = c - 'a';
            if(tr[now].son[ch] == 0)
                tr[now].son[ch] = cnt++;
            now = tr[now].son[ch];
            tr[now].num++;
        }
        tr[now].isend = true;
    }
    int find(string str)
    {
        int now = 0;
        for(auto c: str)
        {
            int ch = c - 'a';
            if(tr[now].son[ch] == 0) return 3;  // 找不到
            now = tr[now].son[ch];
        }
        if(tr[now].isend == false) return 3;    // 不是结尾
        if(tr[now].repeat == false)
        {
```

```
            tr[now].repeat = true;
            return 1;    // 点名第一次出现
        }
        return 2;    // 重复出现
        // return tr[now].num 返回以s为前缀的单词的数量
    }
};
```

计算几何

点与向量

```
#include <bits/stdc++.h>

const double pi = acos(-1.0);    // 圆周率，精确到15位小数
const double eps = 1e-8;         // 偏差值，有时用1e-10，注意精度
int sgn(double x){
    if(fabs(x) < eps) return 0; // x == 0
    else return x < 0 ? -1 : 1;
}
int dcmp(double x, double y){
    if(fabs(x - y) < eps) return 0; // x == y;
    else return x < y ? -1 : 1;
}

struct Point{
    double x, y;
    Point() {}
    Point(double x, double y):x(x), y(y) {}
    Point operator+(Point B) {return Point(x+B.x, y+B.y);}
    Point operator-(Point B) {return Point(x-B.x, y-B.y);}
    Point operator*(double k) {return Point(x*k, y*k);}
    Point operator/(double k) {return Point(x/k, y/k);}
    bool operator==(Point B) {return sgn(x-B.x) == 0 && sgn(y-B.y)==0;}
};

double Distance(Point A, Point B) {return hypot(A.x-B.x, A.y-B.y);}

typedef Point Vector;
double Dot(Vector A, Vector B) {return A.x*B.x + A.y*B.y;}
double Len(Vector A) {return sqrt(Dot(A, A));}
double Len2(Vector A) {return Dot(A, A);}
```

```cpp
double Angle(Vector A, Vector B) {return acos(Dot(A,B)/Len(A)/Len(B));}

double Cross(Vector A, Vector B) {return A.x*B.y - A.y*B.x;}
// A, B, C逆时针为正，三角形除以2
double Area2(Point A, Point B, Point C) {return Cross(B-A, C-A);}
// 逆时针旋转角度rad
Vector Rotate(Vector A, double rad){
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}
// Rotate(A, pi/2): return Vector(-A.y, A.x)
// Rotate(A, -pi/2): return Vector(A.y, -A.x)
// 单位法向量
Vector Normal(Vector A) {return Vector(-A.y/Len(A), A.x/Len(A));}
// 两个向量是否平行或重合
bool Parallel(Vector A, Vector B) {return sgn(Cross(A, B)) == 0;}
```

jiangly模板节选

本部分用于补充上面未出现的部分

一、杂类

01 - int128 输出流自定义

```cpp
using i128 = __int128;

std::ostream &operator<<(std::ostream &os, i128 n) {
    std::string s;
    while (n) {
        s += '0' + n % 10;
        n /= 10;
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

02 - 常用库函数重载

```
using i64 = long long;
using i128 = __int128;

i64 ceilDiv(i64 n, i64 m) {
    if (n >= 0) {
        return (n + m - 1) / m;
    } else {
        return n / m;
    }
}

i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
    } else {
        return (n - m + 1) / m;
    }
}

template<class T>
void chmax(T &a, T b) {
    if (a < b) {
        a = b;
    }
}

i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}
```

二、图与网络

08 - 树链剖分（HLD）

```
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
```

```
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
```

```
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    bool isAncester(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }

    int rootedParent(int u, int v) {
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
```

```
        }
        auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x,
int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

三、数论、几何、多项式

04 - 求解单个数的欧拉函数

```
int phi(int n) {
    int res = n;
    for (int i = 2; i * i ⩽ n; i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res = res / i * (i - 1);
        }
    }
    if (n > 1) {
        res = res / n * (n - 1);
    }
    return res;
```

```
}
```

05 - 扩展欧几里得（exGCD）

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

06 - 组合数（Comb+MInt & MLong）

```
struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = std::min(m, Z::getMod() - 1);
        if (m ⩽ n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i ⩽ m; i++) {
            _fac[i] = _fac[i - 1] * i;
```

```
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;
```

08 - 素数测试与因式分解（Miller-Rabin & Pollard-Rho）

```
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
```

```
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
std::vector<i64> factorize(i64 n) {
    std::vector<i64> p;
    std::function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
        if (isprime(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {
            return (mul(x, x, n) + 1) % n;
        };
        i64 x0 = 2;
        while (true) {
            i64 x = x0;
            i64 y = x0;
            i64 d = 1;
            i64 power = 1, lam = 0;
            i64 v = 1;
            while (d == 1) {
                y = g(y);
                ++lam;
```

```
                v = mul(v, std::abs(x - y), n);
                if (lam % 127 == 0) {
                    d = std::gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = std::gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    std::sort(p.begin(), p.end());
    return p;
}
```

四、数据结构

03A - 线段树（SegmentTree 基础区间加乘）

```
struct SegmentTree {
    int n;
    std::vector<int> tag, sum;
    SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) {}

    void pull(int p) {
        sum[p] = (sum[2 * p] + sum[2 * p + 1]) % P;
    }
```

```
    void mul(int p, int v) {
        tag[p] = 1LL * tag[p] * v % P;
        sum[p] = 1LL * sum[p] * v % P;
    }

    void push(int p) {
        mul(2 * p, tag[p]);
        mul(2 * p + 1, tag[p]);
        tag[p] = 1;
    }

    int query(int p, int l, int r, int x, int y) {
        if (l ⩾ y || r ⩽ x) {
            return 0;
        }
        if (l ⩾ x && r ⩽ y) {
            return sum[p];
        }
        int m = (l + r) / 2;
        push(p);
        return (query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y)) % P;
    }

    int query(int x, int y) {
        return query(1, 0, n, x, y);
    }

    void rangeMul(int p, int l, int r, int x, int y, int v) {
        if (l ⩾ y || r ⩽ x) {
            return;
        }
        if (l ⩾ x && r ⩽ y) {
            return mul(p, v);
        }
        int m = (l + r) / 2;
        push(p);
        rangeMul(2 * p, l, m, x, y, v);
        rangeMul(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeMul(int x, int y, int v) {
        rangeMul(1, 0, n, x, y, v);
    }

    void add(int p, int l, int r, int x, int v) {
        if (r - l == 1) {
```

```
            sum[p] = (sum[p] + v) % P;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            add(2 * p, l, m, x, v);
        } else {
            add(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void add(int x, int v) {
        add(1, 0, n, x, v);
    }
};
```

03B - 线段树（SegmentTree+Info 查找前驱后继）

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
```

```
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l ≥ y || r ≤ x) {
            return Info();
        }
        if (l ≥ x && r ≤ y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l ≥ y || r ≤ x || !pred(info[p])) {
            return -1;
        }
```

```
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
struct Info {
    int cnt = 0;
    i64 sum = 0;
    i64 ans = 0;
};
Info operator+(Info a, Info b) {
    Info c;
    c.cnt = a.cnt + b.cnt;
    c.sum = a.sum + b.sum;
    c.ans = a.ans + b.ans + a.cnt * b.sum - a.sum * b.cnt;
    return c;
}
```

03C - 线段树（SegmentTree+Info+Merge 区间合并）

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
```

```
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l ⩾ y || r ⩽ x) {
            return Info();
        }
        if (l ⩾ x && r ⩽ y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l ⩾ y || r ⩽ x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l ⩾ y || r ⩽ x || !pred(info[p])) {
            return -1;
```

```
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Info {
    int x = 0;
    int cnt = 0;
};

Info operator+(Info a, Info b) {
    if (a.x == b.x) {
        return {a.x, a.cnt + b.cnt};
    } else if (a.cnt > b.cnt) {
        return {a.x, a.cnt - b.cnt};
    } else {
        return {b.x, b.cnt - a.cnt};
    }
}
```

04A - 懒标记线段树（LazySegmentTree 基础区间修改）

```
template<class Info, class Tag>
struct LazySegmentTree {
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 <<
std::__lg(n)) {}
    LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
```

```
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l ≥ y || r ≤ x) {
            return Info();
        }
        if (l ≥ x && r ≤ y) {
            return info[p];
```

```
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l ⩾ y || r ⩽ x) {
            return;
        }
        if (l ⩾ x && r ⩽ y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    void half(int p, int l, int r) {
        if (info[p].act == 0) {
            return;
        }
        if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {
            apply(p, {-(info[p].min + 1) / 2});
            return;
        }
        int m = (l + r) / 2;
        push(p);
        half(2 * p, l, m);
        half(2 * p + 1, m, r);
        pull(p);
    }
    void half() {
        half(1, 0, n);
    }
};

constexpr i64 inf = 1E18;
```

```
struct Tag {
    i64 add = 0;

    void apply(Tag t) {
        add += t.add;
    }
};

struct Info {
    i64 min = inf;
    i64 max = -inf;
    i64 sum = 0;
    i64 act = 0;

    void apply(Tag t) {
        min += t.add;
        max += t.add;
        sum += act * t.add;
    }
};

Info operator+(Info a, Info b) {
    Info c;
    c.min = std::min(a.min, b.min);
    c.max = std::max(a.max, b.max);
    c.sum = a.sum + b.sum;
    c.act = a.act + b.act;
    return c;
}
```

04B - 懒标记线段树（LazySegmentTree 查找前驱后继）

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
```

```
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
```

```
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l ⩾ y || r ⩽ x) {
            return Info();
        }
        if (l ⩾ x && r ⩽ y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l ⩾ y || r ⩽ x) {
            return;
        }
        if (l ⩾ x && r ⩽ y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l ⩾ y || r ⩽ x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
```

```
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Tag {
    i64 a = 0, b = 0;
    void apply(Tag t) {
        a = std::min(a, b + t.a);
        b += t.b;
    }
};

int k;

struct Info {
    i64 x = 0;
    void apply(Tag t) {
        x += t.a;
        if (x < 0) {
```

```
            x = (x % k + k) % k;
        }
        x += t.b - t.a;
    }
};
Info operator+(Info a, Info b) {
    return {a.x + b.x};
}
```

04C - 懒标记线段树（LazySegmentTree 二分修改）

```
constexpr int inf = 1E9 + 1;
template<class Info, class Tag>
struct LazySegmentTree {
    const int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 <<
std::__lg(n)) {}
    LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size()) {
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
```

```
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
```

```
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    void maintainL(int p, int l, int r, int pre) {
        if (info[p].difl > 0 && info[p].maxlowl < pre) {
            return;
        }
        if (r - l == 1) {
            info[p].max = info[p].maxlowl;
            info[p].maxl = info[p].maxr = l;
            info[p].maxlowl = info[p].maxlowr = -inf;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        maintainL(2 * p, l, m, pre);
        pre = std::max(pre, info[2 * p].max);
        maintainL(2 * p + 1, m, r, pre);
        pull(p);
    }
    void maintainL() {
        maintainL(1, 0, n, -1);
    }
    void maintainR(int p, int l, int r, int suf) {
        if (info[p].difr > 0 && info[p].maxlowr < suf) {
            return;
        }
        if (r - l == 1) {
            info[p].max = info[p].maxlowl;
            info[p].maxl = info[p].maxr = l;
            info[p].maxlowl = info[p].maxlowr = -inf;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        maintainR(2 * p + 1, m, r, suf);
        suf = std::max(suf, info[2 * p + 1].max);
        maintainR(2 * p, l, m, suf);
        pull(p);
    }
    void maintainR() {
        maintainR(1, 0, n, -1);
    }
};

struct Tag {
```

```
    int add = 0;

    void apply(Tag t) & {
        add += t.add;
    }
};

struct Info {
    int max = -1;
    int maxl = -1;
    int maxr = -1;
    int difl = inf;
    int difr = inf;
    int maxlowl = -inf;
    int maxlowr = -inf;

    void apply(Tag t) & {
        if (max != -1) {
            max += t.add;
        }
        difl += t.add;
        difr += t.add;
    }
};

Info operator+(Info a, Info b) {
    Info c;
    if (a.max > b.max) {
        c.max = a.max;
        c.maxl = a.maxl;
        c.maxr = a.maxr;
    } else if (a.max < b.max) {
        c.max = b.max;
        c.maxl = b.maxl;
        c.maxr = b.maxr;
    } else {
        c.max = a.max;
        c.maxl = a.maxl;
        c.maxr = b.maxr;
    }

    c.difl = std::min(a.difl, b.difl);
    c.difr = std::min(a.difr, b.difr);
    if (a.max != -1) {
        c.difl = std::min(c.difl, a.max - b.maxlowl);
    }
    if (b.max != -1) {
```

```
        c.difr = std::min(c.difr, b.max - a.maxlowr);
    }

    if (a.max == -1) {
        c.maxlowl = std::max(a.maxlowl, b.maxlowl);
    } else {
        c.maxlowl = a.maxlowl;
    }
    if (b.max == -1) {
        c.maxlowr = std::max(a.maxlowr, b.maxlowr);
    } else {
        c.maxlowr = b.maxlowr;
    }
    return c;
}
```

05B - 取模类（MLong & MInt 新版）

```
template<class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr i64 mul(i64 a, i64 b, i64 p) {
    i64 res = a * b - i64(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template<i64 P>
struct MLong {
    i64 x;
```

```
    constexpr MLong() : x{} {}
    constexpr MLong(i64 x) : x{norm(x % getMod())} {}

    static i64 Mod;
    constexpr static i64 getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(i64 Mod_) {
        Mod = Mod_;
    }
    constexpr i64 norm(i64 x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x ⩾ getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr i64 val() const {
        return x;
    }
    explicit constexpr operator i64() const {
        return x;
    }
    constexpr MLong operator-() const {
        MLong res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MLong inv() const {
        assert(x ≠ 0);
        return power(*this, getMod() - 2);
    }
    constexpr MLong &operator*=(MLong rhs) & {
        x = mul(x, rhs.x, getMod());
        return *this;
    }
    constexpr MLong &operator+=(MLong rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MLong &operator-=(MLong rhs) & {
```

```
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MLong &operator/=(MLong rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MLong operator*(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MLong operator+(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MLong operator-(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MLong operator/(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
        i64 v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MLong
&a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MLong lhs, MLong rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
i64 MLong<0LL>::Mod = i64(1E18) + 9;
```

```cpp
template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
```

```
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a)
{
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};
```

```
template<>
int MInt<0>::Mod = 998244353;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 1000000007;
using Z = MInt<P>;
```

五、字符串

01 - 马拉车（Manacher）

2023-05-14

```
std::vector<int> manacher(std::string s) {
    std::string t = "#";
    for (auto c : s) {
        t += c;
        t += '#';
    }
    int n = t.size();
    std::vector<int> r(n);
    for (int i = 0, j = 0; i < n; i++) {
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = std::min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
            r[i] += 1;
        }
        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
    return r;
}
```

02 - Z函数

2023-08-11

```
std::vector<int> zFunction(std::string s) {
    int n = s.size();
    std::vector<int> z(n + 1);
    z[0] = n;
    for (int i = 1, j = 1; i < n; i++) {
        z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > j + z[j]) {
            j = i;
        }
    }
    return z;
}
```

03 - 后缀数组（SA）

2023-03-14

```
struct SuffixArray {
    int n;
    std::vector<int> sa, rk, lc;
    SuffixArray(const std::string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(), [&](int a, int b) {return s[a] <
s[b];});
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
```

```
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; --i)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            std::swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; ++i)
                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] ||
sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i +
j] == s[sa[rk[i] - 1] + j]; )
                    ++j;
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

04A - 后缀自动机（SuffixAutomaton 旧版）

2022-08-17

```
struct SuffixAutomaton {
    static constexpr int ALPHABET_SIZE = 26, N = 5e5;
    struct Node {
        int len;
        int link;
        int next[ALPHABET_SIZE];
        Node() : len(0), link(0), next{} {}
    } t[2 * N];
    int cntNodes;
    SuffixAutomaton() {
```

```
        cntNodes = 1;
        std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
        t[0].len = -1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1)
                return q;
            int r = ++cntNodes;
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = ++cntNodes;
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
};
```

04B - 后缀自动机（SAM 新版）

2023-05-27

```
struct SAM {
    static constexpr int ALPHABET_SIZE = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, next{} {}
    };
    std::vector<Node> t;
    SAM() {
```

```
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1) {
                return q;
            }
            int r = newNode();
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            t[r].next = t[q].next;
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = newNode();
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
    int extend(int p, char c, char offset = 'a') {
        return extend(p, c - offset);
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
```

```
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

05 - 回文自动机（PAM）

2023-05-19

```
struct PAM {
    static constexpr int ALPHABET_SIZE = 28;
    struct Node {
        int len;
        int link;
        int cnt;
        std::array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, cnt{}, next{} {}
    };
    std::vector<Node> t;
    int suff;
    std::string s;
    PAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].len = -1;
        suff = 1;
        s.clear();
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
```

```
    bool add(char c, char offset = 'a') {
        int pos = s.size();
        s += c;
        int let = c - offset;
        int cur = suff, curlen = 0;

        while (true) {
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
                break;
            cur = t[cur].link;
        }
        if (t[cur].next[let]) {
            suff = t[cur].next[let];
            return false;
        }

        int num = newNode();
        suff = num;
        t[num].len = t[cur].len + 2;
        t[cur].next[let] = num;

        if (t[num].len == 1) {
            t[num].link = 1;
            t[num].cnt = 1;
            return true;
        }

        while (true) {
            cur = t[cur].link;
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                t[num].link = t[cur].next[let];
                break;
            }
        }

        t[num].cnt = 1 + t[t[num].link].cnt;

        return true;
    }
};

PAM pam;
```

06A - AC自动机（AC 旧版）

2021-07-07

```
constexpr int N = 3e5 + 30, A = 26;

struct Node {
    int fail;
    int sum;
    int next[A];
    Node() : fail(-1), sum(0) {
        std::memset(next, -1, sizeof(next));
    }
} node[N];

int cnt = 0;
int bin[N];
int nBin = 0;

int newNode() {
    int p = nBin > 0 ? bin[--nBin] : cnt++;
    node[p] = Node();
    return p;
}

struct AC {
    std::vector<int> x;
    AC(AC &&a) : x(std::move(a.x)) {}
    AC(std::vector<std::string> s, std::vector<int> w) {
        x = {newNode(), newNode()};
        std::fill(node[x[0]].next, node[x[0]].next + A, x[1]);
        node[x[1]].fail = x[0];

        for (int i = 0; i < int(s.size()); i++) {
            int p = x[1];
            for (int j = 0; j < int(s[i].length()); j++) {
                int c = s[i][j] - 'a';
                if (node[p].next[c] == -1) {
                    int u = newNode();
                    x.push_back(u);
                    node[p].next[c] = u;
                }
                p = node[p].next[c];
            }
            node[p].sum += w[i];
        }
```

```
        std::queue<int> que;
        que.push(x[1]);
        while (!que.empty()) {
            int u = que.front();
            que.pop();
            node[u].sum += node[node[u].fail].sum;
            for (int c = 0; c < A; c++) {
                if (node[u].next[c] == -1) {
                    node[u].next[c] = node[node[u].fail].next[c];
                } else {
                    node[node[u].next[c]].fail = node[node[u].fail].next[c];
                    que.push(node[u].next[c]);
                }
            }
        }
    }
    ~AC() {
        for (auto p : x) {
            bin[nBin++] = p;
        }
    }
    i64 query(const std::string &s) const {
        i64 ans = 0;
        int p = x[1];
        for (int i = 0; i < int(s.length()); i++)  {
            int c = s[i] - 'a';
            p = node[p].next[c];
            ans += node[p].sum;
        }
        return ans;
    }
};
```

06B - AC自动机（AhoCorasick 新版）

2023-04-07

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET> next;
        Node() : link{}, next{} {}
```

```
    };

    std::vector<Node> t;

    AhoCorasick() {
        init();
    }

    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }

    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    int add(const std::vector<int> &a) {
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        return p;
    }

    int add(const std::string &a, char offset = 'a') {
        std::vector<int> b(a.size());
        for (int i = 0; i < a.size(); i++) {
            b[i] = a[i] - offset;
        }
        return add(b);
    }

    void work() {
        std::queue<int> q;
        q.push(1);

        while (!q.empty()) {
            int x = q.front();
            q.pop();
```

```
            for (int i = 0; i < ALPHABET; i++) {
                if (t[x].next[i] == 0) {
                    t[x].next[i] = t[t[x].link].next[i];
                } else {
                    t[t[x].next[i]].link = t[t[x].link].next[i];
                    q.push(t[x].next[i]);
                }
            }
        }
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

07 - 随机生成模底 字符串哈希（例题）

2022-06-09

```
#include <bits/stdc++.h>

using i64 = long long;

bool isprime(int n) {
    if (n ⩽ 1) {
        return false;
    }
    for (int i = 2; i * i ⩽ n; i++) {
```

```
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

int findPrime(int n) {
    while (!isprime(n)) {
        n++;
    }
    return n;
}

using Hash = std::array<int, 2>;

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::mt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());

    const int P = findPrime(rng() % 900000000 + 100000000);

    std::string s, x;
    std::cin >> s >> x;

    int n = s.length();
    int m = x.length();

    std::vector<int> h(n + 1), p(n + 1);
    for (int i = 0; i < n; i++) {
        h[i + 1] = (10LL * h[i] + s[i] - '0') % P;
    }
    p[0] = 1;
    for (int i = 0; i < n; i++) {
        p[i + 1] = 10LL * p[i] % P;
    }

    auto get = [&](int l, int r) {
        return (h[r] + 1LL * (P - h[l]) * p[r - l]) % P;
    };

    int px = 0;
    for (auto c : x) {
        px = (10LL * px + c - '0') % P;
```

```
    }

    for (int i = 0; i <= n - 2 * (m - 1); i++) {
        if ((get(i, i + m - 1) + get(i + m - 1, i + 2 * m - 2)) % P == px) {
            std::cout << i + 1 << " " << i + m - 1 << "\n";
            std::cout << i + m << " " << i + 2 * m - 2 << "\n";
            return 0;
        }
    }

    std::vector<int> z(m + 1), f(n + 1);
    z[0] = m;

    for (int i = 1, j = -1; i < m; i++) {
        if (j != -1) {
            z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
        }
        while (z[i] + i < m && x[z[i]] == x[z[i] + i]) {
            z[i]++;
        }
        if (j == -1 || i + z[i] > j + z[j]) {
            j = i;
        }
    }
    for (int i = 0, j = -1; i < n; i++) {
        if (j != -1) {
            f[i] = std::max(0, std::min(j + f[j] - i, z[i - j]));
        }
        while (f[i] + i < n && f[i] < m && x[f[i]] == s[f[i] + i]) {
            f[i]++;
        }
        if (j == -1 || i + f[i] > j + f[j]) {
            j = i;
        }
    }

    for (int i = 0; i + m <= n; i++) {
        int l = std::min(m, f[i]);

        for (auto j : { m - l, m - l - 1 }) {
            if (j <= 0) {
                continue;
            }
            if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
                std::cout << i - j + 1 << " " << i << "\n";
                std::cout << i + 1 << " " << i + m << "\n";
                return 0;
```

```
            }
            if (i + m + j <= n && (get(i, i + m) + get(i + m, i + m + j)) % P
== px) {
                std::cout << i + 1 << " " << i + m << "\n";
                std::cout << i + m + 1 << " " << i + m + j << "\n";
                return 0;
            }
        }
    }

    return 0;
}
```