算法模板

一、数据结构

1 DSU

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

    DSU(int n) {
        init(n);
    }
    void init(int n) {
        fa.resize(n);
        siz.assign(n, 1);
        std::iota(fa.begin(), fa.end(), 0);
    }

    int find(int x) {
        if (x != fa[x]) {
            return fa[x] = find(fa[x]);
        }
        return x;
    }

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

    bool merge(int x, int y) {
        int tx = find(x), ty = find(y);
        if (tx == ty) {
            return false;
        }
        fa[ty] = tx;
        siz[tx] += siz[ty];
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

2 带权DSU

```
i64 fa[size], val[size];
i64 find(i64 x) {
    if (x != fa[x]) {
        i64 t = fa[x];
        fa[x] = find(fa[x]);
        val[x] += val[t];
    }
    return fa[x];
}
```

3 ST表

```
template<typename T>
class ST {
public:
    int n;
    std::vector<T> a;
    std::vector<std::vector<T>> fmin, fmax, fgcd;

    ST(int _n) {
        n = _n;
        a.assign(_n + 1, {});
    };

    void cal_max() {
        fmax.assign(n + 1, std::vector<T>(std::__lg(n) + 1, {}));
        for (int i = 0; i < n; i++) {
            fmax[i][0] = a[i];
        }
        for (int j = 1; j <= std::__lg(n); j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                fmax[i][j] = std::max(fmax[i][j - 1], fmax[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    void cal_min() {
        fmin.assign(n + 1, std::vector<T>(std::__lg(n) + 1, {}));
        for (int i = 0; i < n; i++) {
            fmin[i][0] = a[i];
        }
        for (int j = 1; j <= std::__lg(n); j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                fmin[i][j] = std::min(fmin[i][j - 1], fmin[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    void cal_gcd() {
        fgcd.assign(n + 1, std::vector<T>(std::__lg(n) + 1, {}));
        for (int i = 0; i < n; i++) {
            fgcd[i][0] = a[i];
        }
        for (int j = 1; j <= std::__lg(n); j++) {
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                fgcd[i][j] = std::gcd(fgcd[i][j - 1], fgcd[i + (1 << (j - 1))][j - 1]);
            }
        }

    }

    T get_max(int l, int r) {
        int len = std::__lg(r - l + 1);
        return std::max(fmax[l][len], fmax[r - (1 << len) + 1][len]);
    }

    T get_min(int l, int r) {
        int len = std::__lg(r - l + 1);
        return std::min(fmin[l][len], fmin[r - (1 << len) + 1][len]);
    }

    T get_gcd(int l, int r) {
        int len = std::__lg(r - l + 1);
        return std::gcd(fgcd[l][len], fgcd[r - (1 << len) + 1][len]);
    }
};
```

```
    vector<int> a(n + 1);
    vector<vector<int>> f(n + 1, vector<int> (__lg(n) + 1)); // f[i][j] 表示从第i项往后到第i + (1 << j) - 1项的区间最大值

    // 初始化
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        f[i][0] = a[i];
    }

    for (int j = 1; j <= __lg(n); j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
        }
    }

    while (m--) {
        int l, r;
        cin >> l >> r;

        int len = __lg(r - l + 1);
        cout << max(f[l][len], f[r - (1 << len) + 1][len]) << '\n';
    }
```

4 树状数组

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

    Fenwick(int _n) {
        init(_n);
    }

    void init(int _n) {
        n = _n;
        a.assign(n, T{});
    }

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

    void add(int x, const T& v) {
        for (int i = x; i < n; i += lowbit(i)) {
            a[i] = a[i] + v;
        }
    }

    T sum(int x) {
        T res{};
        for (int i = x; i > 0; i -= lowbit(i)) {
            res = res + a[i];
        }
        return res;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    // 第一个 sum(x) >= k
    int find(const T& k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i > 0; i /= 2) {
            if (x + i <= n && cur + a[x + i] <= k) {
                x += i;
                cur = cur + a[x];
            }
        }
        return x;
    }
};
```

5 (1) 线段树(lazy_tag 区间乘和区间加)

```cpp
i64 mod;
template<class Info, class Tag>
class SegmentTree {
private:
    #define ls(p) (p << 1)
    #define rs(p) (p << 1 | 1)
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

    void pull(int p) {
        info[p] = info[ls(p)] + info[rs(p)];
    }

    void settag(int p, Tag v) {
        info[p].val = ((info[p].val * v.mul % mod) + (info[p].sz * v.add % mod)) % mod;
        tag[p] = tag[p] + v;
    }

    void push(int p) {
        if (tag[p].add || tag[p].mul != 1) {
            settag(ls(p), tag[p]);
            settag(rs(p), tag[p]);
            // 标记下传，消除自身标记
            tag[p] = {};
        }
    }

    void build(int p, int pl, int pr) {
        if (pl == pr) {
            info[p].val = a[pl];
            return;
        }
        int mid = pl + pr >> 1;
        build(ls(p), pl, mid);
        build(rs(p), mid + 1, pr);
        pull(p);
    }

    void rangeModify(int l, int r, int p, int pl, int pr, const Tag v) {
        if (l <= pl && pr <= r) {
            settag(p, v);
            return;
        }
        push(p);
        int mid = pl + pr >> 1;
        if (l <= mid) {
            rangeModify(l, r, ls(p), pl, mid, v);
        }
        if (r > mid) {
            rangeModify(l, r, rs(p), mid + 1, pr, v);
        }
        pull(p);
    }

    Info query(int l, int r, int p, int pl, int pr) {
        if (l <= pl && pr <= r) {
            return info[p];
        }
        push(p);
        int mid = pl + pr >> 1;
        Info res {};
        if (l <= mid) {
            res = res + query(l, r, ls(p), pl, mid);
        }
        if (r > mid) {
            res = res + query(l, r, rs(p), mid + 1, pr);
        }
        return res;
    }
public:
    std::vector<i64> a;
    SegmentTree(int n_) {
        n = n_;
        info.assign(n_ << 2 | 1, {});
        tag.assign(n_ << 2 | 1, {});
        a.assign(n_ + 1, {});
    }
    // 建树
    void build() {
        build(1, 1, n);
    }

    // 区间乘 + 区间加
    void rangeModify(int l, int r, const Tag v) {
        rangeModify(l, r, 1, 1, n, v);
    }

    // 区间查询
    i64 query(int l, int r) {
        Info res = query(l, r, 1, 1, n);
        return res.val;
    }
};
struct Tag {
    i64 add = 0;
    i64 mul = 1;
};
Tag operator + (const Tag& x, const Tag& y) {
    Tag res {};
    res.mul = x.mul * y.mul % mod;
    res.add = ((x.add * y.mul % mod) + y.add) % mod;
    return res;
}
```

```
struct Info {
    i64 val {};
    int sz = 1;
};
Info operator + (const Info&x, const Info& y) {
    Info res {};
    res.val = (x.val + y.val) % mod;
    res.sz = (x.sz + y.sz) % mod;
    return res;
}
```

(2) 线段树单点修改+维护区间最大子段和

```
int a[N];

struct node {
    int t;
    int sum, ans, maxl, maxr, sz;
}segtree[N << 2];

void pull(int p) {
    segtree[p].sum = segtree[p << 1].sum + segtree[p << 1 | 1].sum;
    segtree[p].maxl = std::max(segtree[p << 1].maxl, segtree[p << 1].sum + segtree[p << 1 | 1].maxl);
    segtree[p].maxr = std::max(segtree[p << 1 | 1].maxr, segtree[p << 1 | 1].sum + segtree[p << 1].maxr);
    segtree[p].ans = std::max({segtree[p << 1].ans, segtree[p << 1 | 1].ans, segtree[p << 1].maxr + segtree[p
<< 1 | 1].maxl});
}

void build(int p, int pl, int pr) {
    if (pl == pr) {
        segtree[p].sum = a[pl];
        segtree[p].ans = a[pl];
        segtree[p].maxl = a[pl];
        segtree[p].maxr = a[pl];
    } else {
        int mid = pl + pr >> 1;
        build(p << 1, pl, mid);
        build(p << 1 | 1, mid + 1, pr);
        pull(p);
    }
}

void modify(int pos, int p, int pl, int pr, int t) {
    if (pos == pl && pos == pr) {
        segtree[p].ans = segtree[p].maxl = segtree[p].maxr = segtree[p].sum = t;
        return;
    }
    int mid = pl + pr >> 1;
    if (pos <= mid) modify(pos, p << 1, pl, mid, t);
    else modify(pos, p << 1 | 1, mid + 1, pr, t);
    pull(p);
}

node query(int l, int r, int p, int pl, int pr) {
    if (l <= pl && pr <= r) {
        return segtree[p];
    }
    int mid = pl + pr >> 1;
    node res, lt, rt;
    lt.ans = lt.maxl = lt.maxr = rt.ans = rt.maxl = rt.maxr = -1e9;
    res.sum = lt.sum = rt.sum = 0;
    if (l <= mid) {
        lt = query(l, r, p << 1, pl, mid);
        res.sum += lt.sum;
    }
    if (r > mid) {
        rt = query(l, r, p << 1 | 1, mid + 1, pr);
        res.sum += rt.sum;
    }
    res.maxl = std::max(lt.maxl, lt.sum + rt.maxl);
    res.maxr = std::max(rt.maxr, rt.sum + lt.maxr);
    res.ans = std::max({lt.ans, rt.ans, lt.maxr + rt.maxl});
    return res;
}
```

(3) 线段树区间赋值

```cpp
template<class Info, class Tag>
class SegmentTree {
private:
    #define ls(p) (p << 1)
    #define rs(p) (p << 1 | 1)
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

    void pull(int p) {
        info[p] = info[ls(p)] + info[rs(p)];
    }

    void settag(int p, Tag v) {
        if (v.agn != -2e18) {
            info[p].val = v.agn;
        }
        tag[p] = v;
    }

    void push(int p) {
        if (tag[p].agn != -2e18) {
            settag(ls(p), tag[p]);
            settag(rs(p), tag[p]);
            // 标记下传，消除自身标记
            tag[p].agn = -2e18;
        }
    }

    void build(int p, int pl, int pr) {
        if (pl == pr) {
            info[p].val = a[pl];
            return;
        }
        int mid = pl + pr >> 1;
        build(ls(p), pl, mid);
        build(rs(p), mid + 1, pr);
        pull(p);
    }

    void rangeModify(int l, int r, int p, int pl, int pr, const Tag v) {
        if (l <= pl && pr <= r) {
            settag(p, v);
            return;
        }
        push(p);
        int mid = pl + pr >> 1;
        if (l <= mid) {
            rangeModify(l, r, ls(p), pl, mid, v);
        }
        if (r > mid) {
            rangeModify(l, r, rs(p), mid + 1, pr, v);
        }
        pull(p);
    }

    Info query(int l, int r, int p, int pl, int pr) {
        if (l <= pl && pr <= r) {
            return info[p];
        }
        push(p);
        int mid = pl + pr >> 1;
        Info res {};
        if (l <= mid) {
            res = res + query(l, r, ls(p), pl, mid);
        }
        if (r > mid) {
            res = res + query(l, r, rs(p), mid + 1, pr);
        }
        return res;
    }
public:
    std::vector<i64> a;
    SegmentTree(int n_) {
        n = n_;
        info.assign(n_ << 2 | 1, {});
        tag.assign(n_ << 2 | 1, {});
        a.assign(n_ + 1, {});
    }
    // 建树
    void build() {
        build(1, 1, n);
    }

    // 区间修改
    void rangeModify(int l, int r, const Tag v) {
        rangeModify(l, r, 1, 1, n, v);
    }

    // 区间查询
    i64 query(int l, int r) {
        Info res = query(l, r, 1, 1, n);
        return res.val;
    }
};
struct Tag {
    i64 agn = -2e18;
};
Tag operator + (const Tag& x, const Tag& y) {
    return y;
}

struct Info {
    i64 val {};
    int sz = 1;
};
Info operator + (const Info&x, const Info& y) {
    Info res {};
    res.val = x.val + y.val;
    res.sz = x.sz + y.sz;
    return res;
}
```

(4) 线段树区间加

```
template<class Info, class Tag>
class SegmentTree {
private:
    #define ls(p) (p << 1)
    #define rs(p) (p << 1 | 1)
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

    void pull(int p) {
        info[p] = info[ls(p)] + info[rs(p)];
    }

    void settag(int p, Tag v) {
        info[p].val += v.add * info[p].sz;
        tag[p] = tag[p] + v;
    }

    void push(int p) {
        if (tag[p].add) {
            settag(ls(p), tag[p]);
            settag(rs(p), tag[p]);
            // 标记下传，消除自身标记
            tag[p] = {};
        }
    }

    void build(int p, int pl, int pr) {
        if (pl == pr) {
            info[p].val = a[pl];
            return;
        }
        int mid = pl + pr >> 1;
        build(ls(p), pl, mid);
        build(rs(p), mid + 1, pr);
        pull(p);
    }

    void rangeModify(int l, int r, int p, int pl, int pr, const Tag v) {
        if (l <= pl && pr <= r) {
            settag(p, v);
            return;
        }
        push(p);
        int mid = pl + pr >> 1;
        if (l <= mid) {
            rangeModify(l, r, ls(p), pl, mid, v);
        }
        if (r > mid) {
            rangeModify(l, r, rs(p), mid + 1, pr, v);
        }
        pull(p);
    }

    Info query(int l, int r, int p, int pl, int pr) {
        if (l <= pl && pr <= r) {
            return info[p];
        }
        push(p);
        int mid = pl + pr >> 1;
        Info res {};
        if (l <= mid) {
            res = res + query(l, r, ls(p), pl, mid);
        }
        if (r > mid) {
            res = res + query(l, r, rs(p), mid + 1, pr);
        }
        return res;
    }
public:
    std::vector<i64> a;
    SegmentTree(int n_) {
        n = n_;
        info.assign(n_ << 2 | 1, {});
        tag.assign(n_ << 2 | 1, {});
        a.assign(n_ + 1, {});
    }
    // 建树
    void build() {
        build(1, 1, n);
    }

    // 区间加
    void rangeModify(int l, int r, const Tag v) {
        rangeModify(l, r, 1, 1, n, v);
    }

    // 区间查询
    i64 query(int l, int r) {
        Info res = query(l, r, 1, 1, n);
        return res.val;
    }
};
struct Tag {
    i64 add {};
};
Tag operator + (const Tag& x, const Tag& y) {
    return {x.add + y.add};
}

struct Info {
    i64 val {};
    int sz = 1;
};
Info operator + (const Info&x, const Info& y) {
    Info res {};
    res.val = x.val + y.val;
    res.sz = x.sz + y.sz;
    return res;
}
```

二、数论

0、基本数据运算方式

```
const int N = 1e5 + 10;
const int mod = 998244353;
std::vector<i64> fac(N + 1, 1), invfac(N + 1, 1);
i64 qmi(i64 a, i64 b) {
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
void init(int n) {
    fac[0] = 1;
    for (int i = 1; i <= n; i++) {
        fac[i] = fac[i - 1] * i % mod;
    }
    invfac[n] = qmi(fac[n], mod - 2);
    for (int i = n - 1; i >= 0; i--) {
        invfac[i] = invfac[i + 1] * (i + 1) % mod;
    }
}
i64 C(int n, int m) {  // 组合数
    if (m > n || m < 0) {
        return 0;
    }
    return fac[n] * invfac[m] % mod * invfac[n - m] % mod;
}
i64 A(int n, int m) {  // 排列数
    if (m > n || m < 0) {
        return 0;
    }
    return fac[n] * invfac[n - m] % mod;
}
i64 catalan(int n) {  // 卡特兰数
    if (n < 0) {
        return 0;
    }
    return C(2 * n, n) * qmi(n + 1, mod - 2) % mod;
}
```

1 裴蜀定理

设$a_1,a_2,\ldots,a_n$是不全为零的整数，则存在整数$x_1,x_2,\ldots,x_n$,使得$a_1x_1+a_2x_2+\cdots+a_nx_n=\gcd(a_1,a_2,\ldots,a_n)$。

逆定理:

设$a_1,a_2,\ldots,a_n$是不全为零的整数，$d>0$是$a_1,a_2,\ldots,a_n$的公因数，若存在整数$x_1,x_2,\ldots,x_n$,使得$a_1x_1+a_2x_2+\cdots+a_nx_n=d$，则$d=\gcd(a_1,a_2,\ldots,$

2、组合数C(n, k) (杨辉三角递推)

```
    vector<vector<i64>> C(size, vector<i64> (size));
    C[0][0] = 1;
    for (int i = 1; i < size; i++) {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++) {
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % q;
        }
    }
```

3、快速幂

```
i64 qmi(i64 a, i64 b, i64 p) {
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}
```

4、线性筛

```
// vis[i] 表示 i 除了 1 以外的最小除数（一定是个素数）
// vis[i] 为 0 表示 i 是素数
// pri 中存储了所有的 素数
constexpr int N = 4e5 + 1;
int vis[N];
std::vector<int> pri;
void sieve(int n) {
    vis[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!vis[i]) {
            pri.push_back(i);
        }
        for (auto j : pri) {
            if (i * j > n) break;
            vis[i * j] = j;
            if (i % j == 0) break;
        }
    }
}
```

5、矩阵运算

```
struct Matrix {
    int n;
    std::vector<std::vector<i64>> m;
    Matrix(int n_) {
        n = n_;
        m.assign(n_ + 1, std::vector<i64> (n_ + 1, 0ll));
    }
};

Matrix operator * (const Matrix& a, const Matrix& b) {
    int n = std::max(a.n, b.n);
    Matrix res(n);

    for (int i = 1; i <= n; i++) { // a 行
        for (int j = 1; j <= n; j++) { // b 列
            for (int k = 1; k <= n; k++) { // a 列, b 行
                res.m[i][j] = (res.m[i][j] + a.m[i][k] * b.m[k][j]) % p;
            }
        }
    }
    return res;
}

Matrix MatrixPow(Matrix a, i64 b) {
    int n = a.n;
    Matrix res(n);
    for (int i = 1; i <= n; i++) {
        res.m[i][i] = 1;
    }
    while (b) {
        if (b & 1) {
            res = res * a;
        }
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

三、杂项

1、归并排序求逆序对

```
int c[size], res;
inline void ms(int l, int r, int t[]) {
    if (l == r) return;
    int mid = l + r >> 1;
    ms(l, mid, t), ms(mid + 1, r, t);
    int p1 = l, p2 = mid + 1, idx = 0;
    while (p1 <= mid && p2 <= r) {
        if (t[p1] <= t[p2]) c[++idx] = t[p1++];
        else {
            res += mid - p1 + 1;
            c[++idx] = t[p2++];
        }
    }
    while (p1 <= mid) c[++idx] = t[p1++];
    while (p2 <= r) c[++idx] = t[p2++];
    for (int i = 1; i <= idx; i++) t[l + i - 1] = c[i];
}
```

2、__int128输入输出(注意不要关闭同步流)

```
__int128 read() {
    char arr[30];
    __int128 res = 0;
    scanf("%s", arr);
    for (int i = 1; i <= strlen(arr); i++) {
        res *= 10;
        res += arr[i]-'0';
    }
    return res;
}
void show (__int128 num) {
    if (num > 9) { show(num / 10); }
    putchar(num % 10 + '0');
}
```

3、异或哈希

```
    std::vector<u64> a(n + 1), pre(n + 1);
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
    }

    u64 max = *std::max_element(a.begin(), a.end());

    std::mt19937_64 rnd(time(0));
    std::vector<u64> code(max + 1); // max是a[i]的最大值
    for (int i = 1; i <= max; i++) {
        code[i] = rnd();
    }

    for (int i = 1; i <= n; i++) {
        pre[i] = pre[i - 1] ^ code[a[i]];
    }
```

4、对拍

```
#!/bin/bash
t=0
while true; do
    let "t = $t + 1"
    printf $t
    printf ":\n"
    ./random > data.txt
    ./solve < data.txt > solve.out
    ./std < data.txt > std.out

    if diff solve.out std.out; then
        printf "AC\n"
    else
        printf "WA\n"
        cat data.txt
        cat std.out
        cat solve.out
        break
    fi
done
```

```
#include <bits/stdc++.h>
int main() {
    int t = 0;

    while (1) {
        std::cout << "test: " << t++ << std::endl;
        system("gen.exe > data.in");
        system("std.exe < data.in > std.out");
        system("solve.exe < data.in > solve.out");

        if (system("fc std.out solve.out > diff.log")) {
            std::cout << "WA" << std::endl;
            break;
        }
        std::cout << "AC" << std::endl;
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
std::string rand_str(const int len, int k)  /*参数为字符串的长度*/
{
    /*初始化*/
    std::string str;                 /*声明用来保存随机字符串的str*/
    char c;                     /*声明字符c，用来保存随机生成的字符*/
    int idx;                    /*用来循环的变量*/
    /*循环向字符串中添加随机生成的字符*/
    for(idx = 0; idx < len; idx++)
    {
        /*rand()%26是取余，余数为0~25加上'a',就是字母a~z,详见asc码表*/
        c = 'a' + rand() % k;
        str.push_back(c);       /*push_back()是string类尾插函数。这里插入随机字符c*/
    }
    return str;                 /*返回生成的随机字符串*/
}
int main() {
    std::mt19937 rnd(time(0));
    return 0;
}
```

四、图论

1、倍增求lca

```
    int t = int(log(n) / log(2)) + 1;
    std::vector<std::vector<int>> f(n + 1, std::vector<int> (t + 1));

    std::vector<int> d(n + 1);
    d[s] = 1;
    std::queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        for (auto y : e[x]) {
            if (d[y]) continue;
            d[y] = d[x] + 1;
            f[y][0] = x;
            for (int i = 1; i <= t; i++) {
                f[y][i] = f[f[y][i - 1]][i - 1];
            }
            q.push(y);
        }
    }

    auto lca = [&](int x, int y) {
        if (d[x] > d[y]) std::swap(x, y);
        for (int i = t; i >= 0; i--) {
            if (d[f[y][i]] >= d[x]) {
                y = f[y][i];
            }
        }
        if (x == y) return x;
        for (int i = t; i >= 0; i--) {
            if (f[x][i] != f[y][i]) {
                x = f[x][i];
                y = f[y][i];
            }
        }
        return f[x][0];
    };
```

2、树的重心

```
int min = inf, center = -1;
std::vector<int> size(n + 1);
auto dfs = [&](auto &&self, int u, int fa) -> void {
    size[u] = 1;
    int max = 0;
    for (auto y : e[u]) {
        if (y == fa) continue;
        self(self, y, u);
        size[u] += size[y];
        max = std::max(max, size[y]);
    }
    max = std::max(max, n - size[u]);
    if (max <= min) {
        if (max == min) {
            center = std::min(center, u);
        } else {
            center = u;
        }
        min = max;
    }
};
```

3、树的直径

```
// dfs 记录路径 （无法处理负权边）
int tar = 0, max = 0;
std::vector<int> pre(n + 1);
auto dfs = [&](auto &&self, int u, int fa, int w, int tag) -> void {
    if (w > max) {
        max = w;
        tar = u;
    }
    for (auto [y, ww] : e[u]) {
        if (y == fa) continue;
        if (tag == 1) {
            pre[y] = u;
        }
        self(self, y, u, w + ww, tag);
    }
};
dfs(dfs, 1, -1, 0, 0);
int p = tar;
tar = 0, max = 0;
dfs(dfs, p, -1, 0, 1);
int q = tar;

// 树形dp
int ans = -inf;
std::vector<int> dis(n + 1);
auto dp = [&](auto &&self, int u, int fa) -> void {
    for (auto [y, w] : e[u]) {
        if (y == fa) continue;
        self(self, y, u);
        ans = std::max(ans, dis[y] + dis[u] + w);
        dis[u] = std::max(dis[u], dis[y] + w);
    }
};
dp(dp, 1, -1);
```

4、二分图最大匹配（匈牙利算法）$O(nm)$

```
std::vector<int> vis(n + 1), v(m + 1); // v[y] 表示 y 的匹配，vis[u] 表示 u 是否被访问过
auto find = [&](auto &&self, int u) -> bool {
    vis[u] = 1;
    for (auto y : e[u]) {
        if (!v[y] || (!vis[v[y]] && self(self, v[y]))) {
            v[y] = u;
            return true;
        }
    }
    return false;
};
auto match = [&](int x) {
    int res = 0;
    v.assign(m + 1, 0);
```

```
        for (int i = 1; i <= x; i++) {
            vis.assign(n + 1, 0);
            if (find(find, i)) {
                res++;
            }
        }
        return res;
    };
```

5、Dijkstra

```
template<typename T>
struct Dijkstra {
    struct Node {
        int u;
        T w;
        bool operator < (const Node& t) const {
            return w > t.w;
        }
    };

    const int inf = 2e9;
    int n;
    Dijkstra() {}
    Dijkstra(int n) {
        init(n); // 从 0 开始存储
    }

    std::vector<std::vector<std::pair<int, T>>> adj; // 邻接表存图
    std::vector<T> dis;               // 距离
    // 初始化
    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dis.assign(n, inf);          // 初始化为无穷大
    }

    // 加边 u v是边的顶点，w是边权
    void addEdge(int u, int v, T w) {
        adj[u].push_back({v, w});
        // adj[v].push_back({u, w});
    }

    // 单源非负权最短路 s是源
    void shortest_path(int s) {
        std::vector<bool> vis(this->n);
        // 堆优化
        std::priority_queue<Node> pq;
        pq.push({s, 0});
        dis[s] = 0;

        while (!pq.empty()) {
            int u = pq.top().u;
            pq.pop();
            if (vis[u]) continue;
            vis[u] = true;
            for (auto [y, w] : adj[u]) {
                if (dis[y] > dis[u] + w) {
                    dis[y] = dis[u] + w;
                    pq.push({y, dis[y]});
                }
            }
        }
        // dis 已被更新
    }
};
```

五、DP

1、数位 DP

```
i64 dp[14], ten[14]; // dp[i] 表示为 i 位数时每种数字有多少个，ten[i] 表示 10^i
void cal(i64 x, std::vector<i64>& cnt) {
    std::vector<int> num(1);
    while (x) {
        num.push_back(x % 10);
        x /= 10;
    }
    // 299
    for (int i = int(num.size()) - 1; i >= 1; i--) {
        // [00, 99]
        for (int j = 0; j <= 9; j++) {
            cnt[j] += dp[i - 1] * num[i];
        }
        // [000, 200) 中的 0 和 1
        for (int j = 0; j < num[i]; j++) {
            cnt[j] += ten[i - 1];
        }
        i64 num2 = 0;
        for (int j = i - 1; j >= 1; j--) {
            num2 = num2 * 10 + num[j];
        }
        // num2: 99 计算 2 在百位出现的次数
        cnt[num[i]] += num2 + 1; // cnt[2] += 99 + 1
        // 去除前导零 [00, 99]
        cnt[0] -= ten[i - 1]; // cnt[0] -= ten[3 - 1] = cnt[0] - 100
    }
}
void init() {
    ten[0] = 1;
    for (int i = 1; i <= 14; i++) {
        dp[i] = i * ten[i - 1];
        ten[i] = 10 * ten[i - 1];
    }
}
```