icpc 算法模板

Catch-22

2022 年 7 月 12 日

目录

1 数学

1.1 求逆元

注意考虑 x 是 mod 倍数的情况

```
ll qpow(ll a, ll b) {
    ll res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

ll inv(ll x) { return qpow(x, mod - 2); }

const int N = 1e6 + 10;
// 线性递推求逆元 [1, n] 的所有数关于 p 的逆元
int inv[N];
void init_inv () {
    int n, p;
    cin >> n >> p;
    inv[0] = 0, inv[1] = 1;
    for (int i = 2; i <= n; i++)
        inv[i] = (ll)(p - p / i) * inv[p % i] % p;//为了保证大于零加了个 p
    for (int i = 1; i<= n; i++)
        cout << inv[i] << endl;

    return 0;
}
```

1.2 扩展欧几里德算法

bezout 定理：设 a, b 为正整数，则关于 x, y 的方程 $ax + by = c$ 有整数解当且仅当 c 是 $gcd(a, b)$ 的倍数。

返回结果：$ax + by = gcd(a, b)$ 的一组解 (x, y)

时间复杂度：$\mathcal{O}(nlogn)$

```
//拓欧解线性同余方程 a*x=b(mod m)
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
int a, b, m, n;

int exgcd(int a, int b, int &x, int &y) {
    if(b == 0) {
        x = 1, y = 0;
        return a;
```

```
    }
    int d = exgcd(b, a % b, y, x);
    y -= a/b * x;
    return d;
}

int main() {
    int x, y;
    cin >> n;
    while(n -- ) {
        cin >> a >> b >> m;
        int d = exgcd(a, m, x, y);  // d = gcd(a, m)s
        if(b % d != 0) puts("impossible");  //bezout 定理: 有解的条件, gcd(a, m) | b
        else printf("%lld\n", (ll)x * (b/d) % m);
    }
    return 0;
}
```

1.3 筛法

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e7 + 10;
int primes[N], cnt;
bool vis[N]; //合数 true
int n, q;
//linear
void get_prime(int n) {
  for(int i = 2; i <= n; i ++) {
      if(!vis[i]) primes[ ++ cnt] = i;
      for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
          vis[i * primes[j]] = 1;
            if(i % primes[j] == 0) break;
      }
    }
}

//about linear :O(nloglogn)
bool isprime[N];
inline void getprime(int n) {
    for (int i = 2; i <= n; i++) isprime[i] = 1;
    for (int i = 2; i <= n; i++) {
        if(isprime[i]) {
            primes[++cnt] = i;
            if((ll)i*i<=n)
            for (int j = i * i; j <= n; j+=i){
                isprime[j] = 0;
            }
```

```
        }
    }
}
```

```
#include <bits/stdc++.h>
using namespace std;

/*phi compute
根据给定 n 计算 phi(n) O(aqrt(n))
核心公式 phi(n) = n*(1-1/p1)*(1 - 1/p2)*...
*/
int get_phi(int n) {
    int res = n;
    for (int i = 2; i <= n / i; i++) {
        if(n % i == 0) {
            res = res / i * (i - 1); // res *= (1 - 1/n)
            while(n % i == 0)   n /= i;
        }
    }
    if(n > 1)  res = res / n * (n - 1);
    return res;
}

using ll = long long;
const int N = 1e6 + 10;

int phi[N], prime[N];
bool vis[N]; //合数 true

void sel_phi(int n) {
    int cnt = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i ++) {
        if(!vis[i])  {
            prime[cnt++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; prime[j] <= n / i; j ++) {
            vis[prime[j] * i] = true;
            if(i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
                phi[prime[j] * i] = phi[i] * (prime[j] - 1);
        }
    }
}
```

1.4 组合数

1. $C_n^m = C_n^{n-m}$
2. $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$
3. $C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$
4. $lucas: \ C_n^m \equiv C_{n \ mod \ p}^{m \ mod \ p} * C_{n/p}^{m/p}$

```cpp
//求组合数的几种方法
//不确定的时候都开 long long
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int mod = 1e9 + 7, N = 1e6 + 10;
//C(a, b) a 上 b 下

/*1. 依照定义 适用于 a, b 很小的时候（几十）*/
int C(ll a, int b) /* a 上 b 下 */{
    if(a < b) return 0;
    int up = 1, down = 1;
    for (ll i = a; i > a - b; i -- ) up = i % mod * up % mod; //up *= i
    for (int j = 1; j <= b; j ++) down = (ll)j * down % mod; // down *= j
    return (ll)up * qpow(down, mod - 2) % mod; //   (up/down)
}

/*2. 递推 杨辉三角  a, b 在 2000 这个数量级 */
//O(N^2) 1e6~1e7
void init()
{
    for (int i = 0; i < N; i ++)
        for (int j = 0; j <= i; j ++)
            if(!j) C[i][j] = 1;
            else C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
}

//最常用
/*3. 预处理 fac[], invfac[]*/
/**
 * //调用 :
 * 1ll * fac[b] * invfac[a] % mod * invfac[b - a] % mod;
 */
// O(N) 1e6 左右 看 N 大小
int fac[N], invfac[N];
void init() {
    fac[0] = 1;
    for (int i = 1; i < N; i ++)  (ll)fac[i] = fac[i - 1]*i% mod;
    invfac[N - 1] = qpow(fac[N - 1], mod - 2);
    for (int i = N - 2; i >= 0; i --)
        invfac[i] = (ll)invfac[i + 1] * (i + 1) % mod;
```

```
}

/*4. lucas 定理 当 a, b 的值特别大 如 1e9 以上...1e18 等 */
int C(int a, int b) {
    int res = 1;
    for (int i = 1, j = a; i <= b; i ++, j --) {
        res = (ll)res * j % p;
        res = (ll)res * binpow(i, p - 2) % p;
    }
    return res;
}

ll lucas(ll a, ll b) {//p 为质 (模) 数
    if(a < p && b < p) return C(a, b);
    return (ll)C(a % p, b % p) * lucas(a / p, b / p) % p;
}
```

1.5 容斥原理

S_i 为有限集，$|S|$ 为 S 的大小（元素个数），则：

$$|\bigcup_{i=1}^{n} S_i| = \sum_{i=1}^{n} |S_i| - \sum_{1\le i<j\le n} |S_i \cap S_j| + \sum_{1\le i<j<k\le n} |S_i \cap S_j \cap S_k| + \cdots + (-1)^{n+1}|S_1 \cap \cdots \cap S_n|$$

1.6 数论分块

考虑和式：$\sum_{i=1}^{n} f(i)\lfloor\frac{n}{i}\rfloor$，由于 $\lfloor\frac{n}{i}\rfloor$ 的值成一个块状分布，故可以一块一块运算。我们先求出 $f(i)$ 的前缀和，每次以 $[l,r]=[l,\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor]$ 为一块分块求出贡献累加到结果中。(常配合莫反使用)
常见转换：

- $\lceil\frac{a}{b}\rceil = \lfloor\frac{a-1}{b} + 1\rfloor$
- $a \ \ mod \ \ b = a - \lfloor\frac{a}{b}\rfloor * b$

```
// for(int i = st;i<=ed;i++) ans += num/i
ll block(ll st, ll ed, ll num) {
  //sum(num/i i in [st,ed])
  ll L = 0, res = 0;
  ed = min(ed, num);
  for (ll i = st; i <= ed; i = L + 1) {
        L = min(ed, num / (num / i)); //该区间的最后一个数
        res += (L - i + 1) * (num / i); //区间 [i,L] 的 num/i 都是一个值
        // res += (s(L) - s(i-1)) * (num/i); //s(i) 为 f(i) 前缀和
    }
  return res;
}
```

1.7 Möbius 反演

1.8 高斯消元

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 110;
const double eps = 1e-6;
int n;
double a[N][N];

int gauss() {
    int c, r;
    for(c = 0, r = 0; c < n; c ++) {
        int t = r;
        for(int i = r; i < n; i ++)//找到首元素最大
            if(fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if(fabs(a[t][c]) < eps) continue;

        for(int i = c; i <= n; i ++) swap(a[t][i], a[r][i]);
        for(int i = n; i >= c; i --) a[r][i] /= a[r][c];
        for(int i = r + 1; i < n; i ++)
            if(fabs(a[i][c]) > eps)
                for(int j = n; j >= c; j --)
                    a[i][j] -= a[r][j] * a[i][c];
        r ++;
    }
    if(r < n) {
        for(int i = r; i < n; i ++)
            if(fabs(a[i][n]) > eps)
                return 2;
        return 1;
    }

    for(int i = n - 1; i >= 0; i --)
        for(int j = i + 1; j < n; j ++)
            a[i][n] -= a[i][j] * a[j][n];

    return 0;//有唯一解
}

int main() {
    cin >> n ;
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < n + 1; j ++)
            cin >> a[i][j];

    int t = gauss();
```

```
    if(t == 0)
        for(int i = 0; i < n; i ++) printf("%.2f\n", a[i][n]);
    else if(t == 1)
        puts("Infinite group solutions");
    else puts("No solution");

    return 0;
}
```

1.9 Miller Rabin 素数测试

```
//loj143  prime test
#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;
using ll = long long;
/*  O(sqrt(n))
bool is_prime(ll x)
{
    if(x < 2) return false;
    for(ll i = 2; i <= x / i; ++i)
        if(x % i == 0) return false;
    return true;
}
*/
//常常是大素数测试，要用到 int128
inline ll qmul(ll a, ll b, ll p) { return (ll)((__int128)a * b % p); }
ll qpow(ll a, ll b, ll p) {
    ll res = 1;
    while(b) {
        if(b & 1) res = qmul(res, a, p);
        a = qmul(a, a, p);
        b >>= 1;
    }
    return res;
}
const int test_time = 8;

bool mr_test(ll n) {
    if(n < 3 || n % 2 == 0)  return n == 2;
    ll a = n - 1, b = 0;
    while(a % 2 == 0)  a /= 2, ++b;

    for (int i = 1, j; i <= test_time; ++i)  {
        ll x = rand() % (n - 2) + 2, v = qpow(x, a, n);
        if(v == 1)  continue;
        for (j = 0; j < b; ++j) {
            if(v == n - 1) break;
            v = qmul(v, v, n);
```

```
        }
        if(j >= b) return 0;
    }
    return 1;
}

int main() {
    srand(time(0));
    ll x;
    while(cin >> x) {
        if(mr_test(x)) puts("Y");
        else puts("N");
    }
    return 0;
}
```

2 数据结构

2.1 (带权) 并查集

```
const int N = 1e5 + 10;
int fa[N], n, m, d[N];

int find(int x) {return x == fa[x] ? x : fa[x] = find(fa[x]);}
// 对于带权并查集，一般的 find 函数写作:
int find(int x) {
    if(x == fa[x]) return x;
    int rt = find(fa[x]); //这和下面一行顺序很重要
    d[x] += d[fa[x]]; //可以改成 d[x] ^= d[fa[x]], 根据权值意义的需要修改
    return fa[x] = rt;
}

void init() {
    for (int i = 1; i <= n; i++) fa[i] = i;
}
```

2.2 Sparse Table

时间复杂度 $\mathcal{O}(1)$，空间复杂度 $\mathcal{O}(nlogn)$

静态区间查询可重复贡献信息，如“区间最值”、“区间按位和”、“区间按位或”、“区间 GCD”

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5 + 10;
int f[N][21], n, m;
int a[N];
//f[i][j] 表示闭区间 [i, i + 2^j - 1] 的最大值

```

```cpp
void init_st() {
    // cout << __lg(N) << endl;
    for (int j = 0; j < 21; j ++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)//区间长度是 2^j 所以要减一
            if(!j)  f[i][j] = a[i];
            else
                f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
}

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

2.3 树状数组

2.4 线段树

2.5 可持久化线段树

2.6 左偏树

支持操作 (以维护最小值为例):

1. 找到最小值 $\mathcal{O}(1)$
2. 删除最小值 $\mathcal{O}(logn)$
3. 插入一个值 $\mathcal{O}(logn)$
4. 合并两个堆 $\mathcal{O}(logn)$

```cpp
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;
const int N = 2e5 + 10;
int val[N], lson[N], rson[N], dis[N];
int fa[N], idx, n;
int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }

bool cmp(int x, int y) { return val[x] == val[y] ? x < y : val[x] < val[y]; }

int merge(int x, int y) {
    if(!x || !y) return x + y;
    if(cmp(y, x)) swap(x, y);
    rson[x] = merge(rson[x], y);
    if(dis[rson[x]] > dis[lson[x]]) swap(lson[x], rson[x]);
    dis[x] = dis[rson[x]] + 1;
    return x;
}

```

```cpp
int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n;
    val[0] = 2e9;
    while(n --) {
        int op, x, y;  cin >> op;
        if(op == 1) {
            cin >> x;
            val[++idx] = x;
            fa[idx] = idx;
            dis[idx] = 1;
        }
        else if(op == 2) {
            cin >> x >> y;
            x = find(x), y = find(y);
            if(x != y) {
                if(cmp(y, x)) swap(x, y); //x 为较小的
                fa[y] = x;
                merge(x, y);
            }
        }
        else if(op == 3) {
            cin >> x;
            cout << val[find(x)] << endl;
        }
        else { // 删除 x 所在堆的最小值
            cin >> x; x = find(x);
            if(cmp(rson[x], lson[x])) swap(lson[x], rson[x]);
            fa[x] = lson[x], fa[lson[x]] = lson[x];
            merge(lson[x], rson[x]);
        }
    }
    return 0;
}
```

3 图论

3.1 lca

```cpp
/*
求 lca: 1. 倍增  2. 树剖  3.tarjan 离线

lca 用处
1. 树上两点之间的距离 (多维护一个 dist 数组, dis[u] + dis[v] - 2 * dis[lca(u, v)])
2. 树上两条路径是否相交 (如果两条路径相交,那么一定有一条路径的 LCA 在另一条路径上)
*/

//acwing1171 树上距离
#include <bits/stdc++.h>
```

```cpp
#define pb push_back
#define endl '\n'
using namespace std;
const int N = 1e4 + 10;

struct node{int v, w;};
vector<node> G[N];
int fa[N][19], dep[N], dis[N];
int n, m;

void bfs(int s) {
    memset(dep, 0x3f, sizeof dep);
    dep[0] = 0, dep[s] = 1;
    dis[s] = 0;
    queue<int> q; q.push(s);
    while(q.size()) {
        int u = q.front(); q.pop();
        for(auto [v, w] : G[u]) {
            if(dep[v] > dep[u] + 1) {
                dis[v] = dis[u] + w;
                dep[v] = dep[u] + 1;
                fa[v][0] = u;
                q.push(v);
                for(int i = 1; i < 19; ++i)
                    fa[v][i] = fa[fa[v][i - 1]][i - 1];
            }
        }
    }
}

int lca(int a, int b) {
    if(dep[a] < dep[b]) swap(a, b);
    for(int k = 18; k >= 0; k--)
        if(dep[fa[a][k]] >= dep[b])
            a = fa[a][k];
    if(a == b) return a;

    for(int k = 18; k >= 0; --k)
        if(fa[a][k] != fa[b][k])
            a = fa[a][k], b = fa[b][k];
    return fa[a][0];
}

int main() {
    ios::sync_with_stdio(false), cin.tie(0);
    cin >> n >> m;
    for(int i = 1; i < n ; i ++) {
        int u, v, w;  cin >> u >> v >> w;
        G[u].pb({v, w}), G[v].pb({u, w});
```

```
    }

    bfs(1);

    while(m -- ) {
        int u, v; cin >> u >> v;
        int anc = lca(u, v);
        cout << dis[u] + dis[v] - 2 * dis[anc] << endl;
    }
    return 0;
}
```

4 动态规划

4.1 换根 dp

换根 dp 一般时间复杂度为 $\mathcal{O}(n)$，需要对树处理得到大规模答案，如对每个点得到一个答案。

```
// 求树上 对某个点来说包含他的连通点集个数
#include <bits/stdc++.h>
#define pb push_back
#define endl '\n'
using ll = long long;
using namespace std;
const int N = 1e6 + 10, mod = 1e9 + 7;

ll f[N], ans[N], n;
vector<int> G[N];

ll qpow(ll a, ll b) {
    ll res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

void dfs(int u, int fa) {
    f[u] = 1;
    for (auto v:G[u]) {
        if(v == fa) continue;
        dfs(v, u);
        f[u] = f[u] * (f[v] + 1) % mod;
    }
}

/*
```

```
考虑换根，ans[u] 记为以 u 为根，和整棵树其他点能形成的所有子树数量。(即最终答案)
换根方程：ans[v]=( ans[u]/(f[v]+1) +1)*f[v]
解释：u 点答案除以 v 点贡献 (f[v] + 1) 为与 v 无关的 u 点答案，+1 后为其余点对 v 点贡献，再乘上 f[v]

有一个很坑的地方，就是 (f[v]+1) 求逆元可能得到 0 (f[v] 可能为 mod-1)，这时相当于除于 0，出错
当逆元 inv 为 0 时，ans[u] 实际是由在树形 dp 的时候求出的 f[u]，而 f[u] 又等于 (他所有儿子 f 的值 +1) 的乘积。
所以 ans[u] / (f[v]+1) 又可以变成 u 的其他儿子的乘积：u 除 v 外的其他儿子记 brother。
(f[brother_1]+1) * (f[brother_2] + 1) * ..... 他的所有兄弟的值乘积。
*/

void dp(int u, int fa) {
    for (int v:G[u]) {
        if(v == fa) continue;
        ll inv = qpow(f[v] + 1, mod - 2);
        if(inv) ans[v] = (ans[u] * inv % mod + 1) % mod * f[v] % mod;
        else {
            ll t = 1;
            for (auto other:G[u]) {
                if(other == v || other == fa) continue;
                t = t * (f[other] + 1) % mod;
            }
            ans[v] = (t + 1) * f[v] % mod;
        }
        dp(v, u);
    }
}

int main() {
    cin >> n;
    for (int i = 1; i < n; i ++) {
        int u, v; cin >> u >> v;
        G[u].pb(v), G[v].pb(u);
    }
    dfs(1, 0);
    ans[1] = f[1];
    dp(1, 0);

    for (int i = 1; i <= n; i ++) cout << ans[i] << endl;
    return 0;
}
```

5 字符串

5.1 kmp

5.2 title

6 其他

6.1 glibc 内置函数

```cpp
// Returns the number of 1-bits in x.
int __builtin_popcount(unsigned int x);

// Returns the number of trailing 0 (undefined when x == 0)
int __builtin_ctz(unsigned int x);

// Returns log_2(x)
int __lg(int x);

int __gcd(int x, int y);
```

6.2 __int128 读写

```cpp
inline __int128 read(){
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch<'0' || ch>'9') { if(ch == '-') f = -1; ch = getchar();}
    while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
    return x * f;
}

inline void print(__int128 x) {
    if(x < 0) { putchar('-'); x = -x; }
    if(x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```