ICPC 模板

第零部分 引言

这份模板中的代码必须满足：

- 它可以在 C++ 17 标准下运行。
- 模板中的代码必须尽可能简单，函数、变量的命名不能过于冗长，以避免抄写时的麻烦。
- 默认选手使用了 `#include<bits/stdc++.h>` ，所以一般情况下，不应该引用头文件。
- 默认选手使用了 `using namespace std;` ，所以不应该添加 `std::` 前缀。
- 默认选手使用了 `using ll = long long;` ，所以可以使用 `ll` 代替 long long。
- 模板中的代码，尽可能封装成函数或类。但是，如果将一道经典例题的 AC 代码放入模板中，则不受这一点限制。
- 模板中的代码，如果针对于一个算法，则必须说明 **模板的使用方法（或算法功能，或使用例）** 。如果针对一道经典例题，则必须 **简述题意**。

这份模板中的代码**不需要**满足：

- 严格的代码规范。

程序的模板为：

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
void solve() {

}
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    //int t; cin >> t; while(t--)
    solve();
    return 0;
}
```

第一部分 常用算法和数据结构

二分查找

```cpp
using ll = long long;

ll lower(ll l, ll r, ll target, function< ll(ll) > f) {
    if(f(r) < target) return r + 1;
    while (l < r) {
        ll mid = (l + r) / 2;
        if (f(mid) < target)
            l = mid + 1;
        else
            r = mid;
    }
    return l;
}
```

> **功能：在 [l, r] 范围内，求最小的 x 使得 f(x) >= target**
>
> **例子：求 [1, 10^9] 中最小的数 x， 使得递增函数 $x^2 + 5x$ 的值达到或超过给定的数 K**

```
    long long val = lower(1, 1e9, K, [](ll x){
     return x * (x + 5);
    });
```

Mint 自动取模

```
template<int mod> class mint {
public:
    unsigned int x = 0;
    // int get_modular() { return mod; }
    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; }
};
```

> **例子：读入 x 和 y，计算 (3x+5) / y，对 mod = 998244353 取模。**
>
> ```
> Mint x, y;
> cin >> x >> y;
> cout << (3 * x + 5) / y;
> ```

ST 表

```
template <typename T>
struct ST {
    ST(T a[], int n) {
        int t = __lg(n) + 1;
        maxv.resize(t); minv.resize(t);

        for(int i = 0; i < t; i++) maxv[i].resize(n + 1), minv[i].resize(n +
1);
        for(int i = 1; i <= n; i++) maxv[0][i] = minv[0][i] = a[i];

        for (int j = 1; j < t; j++)
            for (int i = 1; i <= n - (1 << j) + 1; i++) {
                maxv[j][i] = max(maxv[j - 1][i], maxv[j - 1][i + (1 << (j -
1))]);
                minv[j][i] = min(minv[j - 1][i], minv[j - 1][i + (1 << (j -
1))]);
            }
    }
    T getmax(int l, int r) {
        int k = __lg(r - l + 1);
        return max(maxv[k][l], maxv[k][r - (1 << k) + 1]);
    }
```

```cpp
    T getmin(int l, int r) {
        int k = __lg(r - l + 1);
        return min(minv[k][l], minv[k][r - (1 << k) + 1]);
    }
private:
    vector<vector<T>> maxv, minv;
};
```

> **例子：对于一个大小为 n， 1-index 的数组 a 建立 ST 表，然后求第 4 个元素至第 8 个元素的最小值**
>
> ```cpp
> ST<int> st(a, n);
> cout << st.getmin(4, 8);
> ```

并查集

```cpp
struct dsu {
    vector<int> p;
    dsu(int n) { p.resize(n + 1); for(int i = 1; i <= n; i++) p[i] = i; }
    int find(int x) { if(x != p[x]) p[x] = find(p[x]); return p[x]; }
    void merge(int x, int y) { p[find(x)] = find(y); }
};
```

> **例子：建立一个并查集，连接 (1， 3) 和 (1， 2) 两条边，并查询 2 和 3 是否在同一集合**
>
> ```cpp
> dsu d(0721);
> d.merge(1, 3);
> d.merge(1, 2);
> assert(d.find(2) == d.find(3));
> ```
>
> **拓展：并查集可以实时查询每个集合的信息（例如集合大小），需要对 merge 函数进行一些处理。**
>
> **使用并查集，连接 (1， 3) 和 (1， 2) 两条边，然后查询 1 所在集合的大小**

```
struct dsu {
    vector<int> p, s;
    dsu(int n) { p.resize(n + 1); s.resize(n + 1); for(int i = 1; i <=
n; i++) p[i] = i, s[i] = 1; }
    int find(int x) { if(x != p[x]) p[x] = find(p[x]); return p[x]; }
    void merge(int x, int y) { x = find(x); y = find(y); if(x != y) p[x]
= y, s[y] += s[x]; }
};

dsu d(0721);
d.merge(1, 3);
d.merge(1, 2);
assert(d.s[d.find(1)] == 3);
```

红黑树

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
typedef
tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update>
rbtree;
```

比起 std::set，它更支持排名的查询。

用法

```
T.insert(x) 插入
T.erase(x) 删除
T.order_of_key(x) 求排名(比它小的元素个数)
T.find_by_order(k) 找排名为 k 的元素的迭代器
T.lower_bound / upper_bound(x) 找大于(等于) x 的迭代器
```

例子：红黑树的插入和查询

```
    rbtree rbt;
    rbt.insert(1);
    rbt.insert(2);
    rbt.insert(3);
    cout << rbt.order_of_key(2) << endl; // 查询比 2 小的元素个数，即 1 个
    cout << *rbt.find_by_order(2) << endl; // 查询排名为 2 的元素，即 3。
```

第二部分 图论

最短路

```
struct WeightedGraph{
    int n;
    int root = 0;

    WeightedGraph(int n): n(n), adj(n + 1) { }

    void add_edge(int u, int v, int dis = 0) {
        adj[u].push_back( {v, dis} );
    }

    /*void read_edges(int m, int read_weight = true) {
        for(int i = 1; i <= m; i++) {
            int u, v, w = 1;
            std::cin >> u >> v;
            if(read_weight) std::cin >> w;
            add_edge(u, v, w);
        }
    }*/

    vector<long long> dijkstra(int start) {
        priority_queue<array<long long, 2>, vector<array<long long, 2>>,
greater<array<long long, 2 >>> pq;
        vector<long long> dist(n + 1, 1e18);
        vector<long long> vis(n + 1, 0);
        dist[start] = 0;
        pq.push( {0, start} );

        while(!pq.empty()) {
            auto [disx, from] = pq.top(); pq.pop();
```

```
            if(vis[from]) continue;
            vis[from] = 1;

            for(auto [to, dis]: adj[from]) {
                if(disx + dis < dist[to]) {
                    dist[to] = disx + dis;
                    pq.push( {dist[to], to} );
                }
            }
        }
        return dist;
    }

    vector<vector<array<long long, 2>>> adj;
};
```

> **功能：堆优化 djikstra 求最短路，时间复杂度是 $\Theta((n+m)\log n)$**
>
> **例子：(P4779) 建图，然后求以 s 为源点的最短路**
>
> ```
> int n, m, s; cin >> n >> m >> s;
> WeightedGraph G(n);
> G.read_edges(m);
> auto v = G.dijkstra(s);
> for(int i = 1; i <= n; i++) cout << v[i] << " ";
> ```

最近公共祖先（倍增法）

```
struct LCA{
    int n;
    int root = 1;

    LCA(int n): n(n), adj(n+1) { }

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    /*void read_edges(){
```

```cpp
        for(int i = 1; i < n; i++) {
            int u, v;
            std::cin >> u >> v;
            add_edge(u, v);
        }
    }*/

    int lca(int u, int v) {
        static vector<array<int, 20>> pa(n + 1);
        static bool prepared = false;
        if(!prepared) {
            depth.resize(n + 1);
            function<void(int, int, int)> dfs = [&](int now, int fa, int d) {
                pa[now][0] = fa;
                depth[now] = d;
                for(auto it: adj[now]) {
                    if(it == fa) continue;
                    dfs(it, now, d + 1);
                }
            };
            dfs(root, root, 0);
            for(int i = 1; i < 20; i++)
                for(int j = 1; j <= n; j++)
                    pa[j][i] = pa[pa[j][i-1]][i-1];
            prepared = true;
        }

        if(depth[u] < depth[v]) swap(u, v);

        for(int i = 19; depth[u] > depth[v]; i--)
            if(depth[pa[u][i]] >= depth[v])
                u = pa[u][i];
        // 将 u 和 v 放到同一高度

        if(u == v) return u;

        for(int i = 19; i >= 0; i--)
            if(pa[u][i] != pa[v][i]) {
                u = pa[u][i];
                v = pa[v][i];
            }

        return pa[u][0];
    }
private:
    vector<vector<int>> adj;
```

```
    vector<int> depth;
};
```

说明：预处理 $\Theta(n \log n)$，单次查询 $\Theta(\log n)$

例子：(P3379) 建树，然后多次查询 LCA

```
LCA t(n);
t.root = s; // 指定根结点，一般来说是 1
t.read_edges();
while(m--) {
 int u, v; cin >> u >> v;
 cout << t.lca(u, v) << '\n';
}
```

第三部分 字符串

KMP 字符串匹配

```
vector<int> get_next(string s){
    // next[i] 表示 s[0..i-1] 的最长公共真前缀后缀长度
    // 返回的长度是 n，如果需要整个串的border，需要 s += '&';
    // "123123" -> [-1 0 0 0 1 2]
    // s += '&';
    int n = s.length();
    vector<int> next(n);
    next[0] = -1;
    for(int i = 1; i < n; i++)
    {
        int val = next[i-1];
        while(val != -1 && s[val] != s[i-1]) val = next[val];
        next[i] = val + 1;
    }
    return next;
};

vector<int> find(const string& s, string t, const vector<int>& next) {
    // 在 s 中找 t 的所有出现，返回 vector
    // next 需要传入 get_next(t)
    // 如果是 1-index，答案需要 + 1
```

```
    int n = s.length();
    int m = t.length();
    vector<int> res;
    int j = -1;
    for(int i = 0; i < n; i++) {
        // 第 j 位失配, 回到第 next[j] 位
        while(j != -1 && s[i] != t[j + 1]) j = next[j];
        if(s[i] == t[j+1]) j++;
        if(j == m - 1) res.push_back(i - m + 1), j = next[j];
    }
    return res;
}
```

> **例子：求 s 中 t 的所有出现，然后输出所有下标(1-index)**
>
> ```
>     string s, t; cin >> s >> t;
>     for(auto it : find(s, t, get_next(t))) cout << it + 1 << '\n';
> ```

第四部分 数据结构

线段树

```
template < typename T,
           T (*op)(T, T) op,
           T e,
           typename F,
           T (*mapping)(F, T, int, int),
           F (*composition)(F, F),
           F e1 >
class segtree
{
    int          n;
    vector< T > v;
    vector< F > lazy;
    void         pushup(int k)
    {
        v[k] = op(v[k * 2], v[k * 2 + 1]);
    }
    void pushdown(int k, int l, int r)
```

```
    {
        v[k] = mapping(v[k], lazy[k], l, r);
        if(l != r)
        {
            lazy[k * 2] = composition(lazy[k * 2], lazy[k]);
            lazy[k * 2 + 1] = composition(lazy[k * 2 + 1], lazy[k]);
        }
        lazy[k] = e1;
    }

    void modify(int now, int ql, int qr, int l, int r, F x)
    {
        pushdown(now, l, r);
        if(l > qr || r < ql) return;
        if(l >= ql && r <= qr)
        {
            lazy[now] = x;
            pushdown(now, l, r);
            return;
        }
        modify(now * 2, ql, qr, l, (l + r) / 2, x);
        modify(now * 2 + 1, ql, qr, (l + r) / 2 + 1, r, x);
        pushup(now);
    }
    T query(int now, int ql, int qr, int l, int r)
    {
        pushdown(now, l, r);
        if(l > qr || r < ql) return e;
        if(l >= ql && r <= qr) return v[now];
        return op(query(now * 2, ql, qr, l, (l + r) / 2),
                  query(now * 2 + 1, ql, qr, (l + r) / 2 + 1, r));
    }

public:
    segtree(int n)
    {
        this->n = n;
        v = vector< T >(n << 2, e);
        lazy = vector< F >(n << 2, e1);
    }

    void modify(int l, int r, F x)
    {
        modify(1, l, r, 1, n, x);
    }
    T query(int l, int r)
```

```
    {
        return query(1, l, r, 1, n);
    }
};

template < typename T >
T MAX(T x, T y) { if(x > y) return x; else return y; }
template < typename T >
T PLUS(T x, T y) { return x + y; }

/*  Note:
    typename T, auto op, T e,
    typename F, auto mapping, auto composition,
    F e1
*/

typedef segtree< ll, PLUS< ll >, 0,
        ll, [](ll x, ll lazy, int l, int r) { return x + (r - l + 1) * lazy; },
        PLUS< ll >, 0 > segtree_add;
```

> 说明：T，op，e，F，mapping，composition，e1 分别表示线段树中维护的值的类型、加法运算、单位元、线段树中懒标记类型、当前值及懒标记向真实值的映射函数、懒标记的叠加函数，懒标记的单位元。
>
> 例子：使用 `segtree_add` ，完成区间加法和区间查询
>
> ```
> segtree_add s(n);
> s.modify(1, 2, 3); // 将 [1, 2] 加上 3
> cout << s.query(1, 2);
> ```

第五部分 数学

拓展欧几里得

```
ll Exgcd(ll a, ll b, ll& x, ll& y)
{
    if(!b) x = 1, y = 0;
        return a;
    }
    ll d = Exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - (a / b) * y;
    return d;
}
```

> **例子：求 $ax+by=\gcd(a,b)$ 的一组整数解。**
>
> ```
> Exgcd(a, b, x, y);
> ```
>
> 则 x,y 是一组整数解。
>
> 如果要求 $ax+by=c$ 的一组整数解，则必须满足 $\gcd(a,b)|c$，然后 x 和 y 需要乘以 $c/\gcd(a,b)$ 才是答案。
>
> 如果要求 x 是正整数，则 x 增加若干个 $b/\gcd(a,b)$ ，y 减少若干个 $a/\gcd(a,b)$ 依然是答案。

带上下界插板法【公式】

下界插板法： n 个元素分成 m 堆，每堆至少有 x 个。

> 相当于 $n-xm$ 个物品分成 m 堆。答案是
>
> $$\binom{n+m-mx-1}{m-1}$$

上界插板法：n 个元素分成 m 堆，每堆至多有 x 个。

> 答案是

$$\sum_{i=0}^{m}(-1)^i\binom{m}{i}\binom{n+m-i(x+1)-1}{m-1}$$