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

Mint 自动取模

```cpp
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]);
    }
    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

```
ST<int> st(a, n);
cout << st.getmin(4, 8);
```

并查集

```
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

```
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

```cpp
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

```cpp
#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

```cpp
    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

```cpp
vector<ll> dijkstra(vector<vector<array<ll, 2>>> adj, int start)
{
    int n = adj.size();
    priority_queue<array<ll, 2>, vector<array<ll, 2>>, greater<>> pq;
    vector<ll> dist(n+1, (1ll<<31)-1);
    vector<ll> 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((array<ll, 2>){dist[to], to});
            }
        }
    }
    return dist;
}
```

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;

vector<vector<array<ll, 2>>> adj(n + 1);

for(int i = 1; i <= m; i++) {
 int x, y, z;
 cin >> x >> y >> z;
 adj[x].push_back({y, z});
}

auto v = dijkstra(adj, s);
for(int i = 1; i <= n; i++) cout << v[i] << " ";
```

最近公共祖先（倍增法）

```
int lca(vector<vector<int>>& adj, int u, int v){

    const int root = 0;
    int n = adj.size();
    static vector<array<int, 20>> pa(n + 1);
    static vector<int> depth(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];
```

```cpp
            }
        }
        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];
        }
    }

    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];
}
```

> **说明：预处理 $\Theta(n\log n)$。用于一棵树上的 LCA 查询，单次查询 $\Theta(\log n)$**
>
> **例子：(P3379) 建树，然后多次查询 LCA**

```cpp
    vector<vector<int>> adj(n + 1);
    for(int i = 1; i < n; i++) {
     int x, y;
     cin >> x >> y;
     adj[x].push_back(y);
     adj[y].push_back(x);
    }

    while(m--)
    {
     int u, v; cin >> u >> v;
     cout << lca(adj, u, v, s) << '\n';
    }
```

第三部分 字符串

KMP 字符串匹配（洛谷 P 3375）

题意：给出 s 和 t，求 s 中 t 的所有出现，然后输出 t 的所有 border 长度。

```cpp
#include<iostream>
#include<cstring>
#define MAXN 1000010
using namespace std;
int kmp[MAXN];
int la,lb,j;
char a[MAXN],b[MAXN];
int main()
{
    cin>>a+1;
    cin>>b+1;
    la=strlen(a+1);
    lb=strlen(b+1);
    for (int i=2;i<=lb;i++)
    {
       while(j&&b[i]!=b[j+1])
        j=kmp[j];
       if(b[j+1]==b[i])j++;
        kmp[i]=j;
    }
    j=0;
    for(int i=1;i<=la;i++)
```

```
    {
          while(j>0&&b[j+1]!=a[i])
            j=kmp[j];
          if (b[j+1]==a[i])
            j++;
          if (j==lb) {cout<<i-lb+1<<endl;j=kmp[j];}
    }

    for (int i=1;i<=lb;i++)
        cout<<kmp[i]<<" ";
    return 0;
}
```

字符串哈希

```
struct strHash{

    const ll base1 = 93, mod1 = 998244353;
    const ll base2 = 97, mod2 = 1e9 + 7;

    int n;
    vector<ll> p1, p2, h1, h2;
    strHash(string s) : n(s.length()), p1(n + 1), p2(n + 1), h1(n + 1), h2(n +
1)
    {
        p1[0] = p2[0] = 1;
        for(int i = 1; i <= n; i++)
        {
            p1[i] = p1[i - 1] * base1 % mod1;
            p2[i] = p2[i - 1] * base2 % mod2;
            h1[i] = h1[i - 1] * base1 + s[i - 1];
            h2[i] = h2[i - 1] * base2 + s[i - 1];
        }
    }
    array<ll, 2> get_hash(int l,int r)
    {
        ll res1 = ((h1[r] - h1[l - 1] * p1[r - l + 1]) % mod1 + mod1) % mod1,
           res2 = ((h2[r] - h2[l - 1] * p2[r - l + 1]) % mod2 + mod2) % mod2;
        return {res1, res2};
    }
};
```

> 说明：双 hash，不采用自然取模

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;
        }
```

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

【临时】

树上背包上下界优化

一般写法：

```
void dfs(int u,int fa){
    // initialize dp[u][val[u]]/dp[u][val[u]~m]
    for(int v:e[u]){
        if(v==fa)continue;
        dfs(v,u);
        for(int i=m;i>=0;--i)
            for(int j=0;j<=min(m,i);++j)
                // dp[u][i] <- dp[u][i-j],dp[v][j]
    }
}
```

优化后：

```
void dfs(int u,int fa){
    // initialize dp[u][val[u]]/dp[u][val[u]~m]
    for(int v:e[u]){
        if(v==fa)continue;
        dfs(v,u);
```

```
            int _sz=sz[u]+sz[v];
            int L=0,R=min(m,_sz);
            for(int i=R;i>=L;--i){
                int l=0,r=min(m,sz[v]);
                for(int j=l;j<=r;++j)
                    // dp[u][i] <- dp[u][i-j],dp[v][j]
            }
            sz[u]=_sz;
        }
    }
```