Template

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll  = long long;
using dd  = long double;
using vi  = vector<int>;
using vl  = vector<long long>;
using vvi = vector<vector<int>>;
using pii = pair<int, int>;
using pll = pair<long long, long long>;
#define all(x) x.begin(), x.end()
#define eb     emplace_back
#define F      first
#define S      second
const int maxn = 200'000;
const int MOD  = 1e9 + 7;
void solve() {}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int tc = 1;
  cin >> tc; // REMEMBER TO COMMENT OUT
  while (tc--) { solve(); }
  return 0;
}
```

DSU

```cpp
struct DSU {
  vi e;
  DSU(int n) : e(n, -1) {}
  bool sameSet(int a, int b) { return find(a) == find(b); }
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) {
      return false;
    }
    if (e[a] > e[b]) {
      swap(a, b);
    }
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

Dinic

```cpp
struct Dinic {
  struct Edge {
    int to, rev;
    ll cap, orig_cap;
    // how much flow in this edge
    ll flow() const { return max(0LL, orig_cap - cap); }
  };

  int n;
  vector<vector<Edge>> adj;
  vi level, next_edge, q;

  Dinic(int n) : n(n), adj(n), level(n), next_edge(n), q(n) {}

  void add_edge(int from, int to, ll cap, ll rev_cap = 0) {
    Edge fwd = {to, (int)adj[to].size(), cap, cap};
    Edge rev = {from, (int)adj[from].size(), rev_cap, rev_cap};
    adj[from].eb(fwd);
    adj[to].eb(rev);
  }

  bool bfs(int s, int t) {
    fill(all(level), -1);
    int q_head = 0, q_tail = 0;
    q[q_tail++] = s;
    level[s]    = 0;

    while (q_head < q_tail && level[t] == -1) {
      int v = q[q_head++];
      for (const Edge &e : adj[v]) {
        if (e.cap > 0 && level[e.to] == -1) {
          level[e.to] = level[v] + 1;
          q[q_tail++] = e.to;
        }
      }
    }
    return level[t] != -1;
  }

  ll dfs(int v, int t, ll pushed) {
    if (v == t || pushed == 0) {
      return pushed;
    }
    for (int &cid = next_edge[v]; cid < (int)adj[v].size(); cid++) {
      Edge &e = adj[v][cid];
      if (e.cap > 0 && level[e.to] == level[v] + 1) {
        ll tr = dfs(e.to, t, min(pushed, e.cap));
        if (tr == 0) {
          continue;
        }
        e.cap -= tr;
        adj[e.to][e.rev].cap += tr;
        return tr;
      }
    }
    return 0;
  }

  ll max_flow(int s, int t) {
    const ll INF = (1LL << 60); // put higher if want
    ll flow      = 0;
    while (bfs(s, t)) {
      fill(all(next_edge), 0);
      while (ll pushed = dfs(s, t, INF)) { flow += pushed; }
    }
    return flow;
  }

  bool is_in_source_side_of_min_cut(int v) const { return level[v] != -1; }
};
```