**Alfred**

# 代码模版库

# 目录

# 1 数据结构

## 1.1 珂朵莉树

支持区间推平，颜色段统计，在随机数据下期望复杂度为 $O(n\log n)$ 的暴力数据结构。

Listing 1: **ChthollyTree.cpp**

```cpp
#include <set>

struct ChthollyTree {
    typedef long long ll;
    struct Node {
        mutable ll l, r, v;
        inline bool operator<(const Node &x) const { return l < x.l; }
    };
    std::set<Node> tr;
    typedef std::set<Node>::iterator iterator;
    ChthollyTree(void) = default;
    ChthollyTree(int rng, int val) { init(rng, val); }
    inline void init(ll rng, ll val) noexcept {
        tr.insert({1, rng, val}), tr.insert({rng + 1, rng + 1, 0});
    }
    inline iterator begin(void) const noexcept { return tr.begin(); }
    inline iterator end(void) const noexcept { return tr.end(); }
    inline iterator split(ll pos) {
        auto it = tr.lower_bound({pos, 0, 0});
        if (it != tr.end() && it->l == pos) return it;
        ll l = (--it)->l, r = it->r, v = it->v;
        tr.erase(it), tr.insert({l, pos - 1, v});
        return tr.insert({pos, r, v}).first;
    }
    inline void assign(ll l, ll r, ll v) {
        auto R = split(r + 1), L = split(l);
        tr.erase(L, R), tr.insert({l, r, v});
    }
    template <class _Functor> // func(iterator)
    inline void modify(ll l, ll r, _Functor func) {
        auto R = split(r + 1), L = split(l);
        for (auto it = L; it != R; it++) func(it);
    }
    template <class _Functor> // func(ll &, iterator)
    inline ll query(ll l, ll r, _Functor func) {
        ll ans = 0;
        auto R = split(r + 1);
        for (auto it = split(l); it != R; it++) func(ans, it);
        return ans;
    }
};
```

## 1.2 树状数组

维护满足结合律且可差分信息的，常数较小的数据结构。

Listing 2: **Fenwick.cpp**

```cpp
#include <vector>

template <class T>
struct Fenwick {
    std::vector<T> c;
    inline int lowbit(int x) { return x & -x; }
    inline void merge(T &x, T y) { x = x + y; } // remember to modify
    inline T subtract(T x, T y) { return x - y; }
    inline void update(size_t pos, T x) {
        for (pos++; pos < c.size(); pos += lowbit(pos)) merge(c[pos], x);
    }
    inline void clear(void) {
        for (auto &x : c) x = T();
    }
    inline T query(size_t pos) {
        T ans = T();
        for (pos++; pos; pos ^= lowbit(pos)) merge(ans, c[pos]);
        return ans;
    }
    inline T query(size_t l, size_t r) {
        return subtract(query(r), query(l - 1));
    }
    Fenwick(size_t len) : c(len + 2) {}
};
```

# 2 比赛配置 and 奇技淫巧

## 2.1 多组数据代码模板

Listing 3: **Template.cpp**

```cpp
#include <bits/stdc++.h>
using namespace std;
using i64 = long long;
const i64 N = 1e5 + 10;
int t = 1;
inline void solve(int Case) {
    // your code here;
}
inline void optimizeIO(void) {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);
}
inline void init(void) {}
int main(int argc, char const *argv[]) {
    optimizeIO(), init(), cin >> t;
    for (int i = 1; i <= t; i++) solve(i);
    return 0;
}
```

## 2.2 快读快写

Listing 4: **FastIO.cpp**

```cpp
namespace fastIO {
    char c, f, e = 0;
    namespace usr {
        template <class _Tp>
        inline int read(_Tp &x) {
            x = f = 0, c = getchar();
            while (!isdigit(c) && !e) f = c == '-', e |= c == EOF, c = getchar();
            while (isdigit(c) && !e) x = (x << 1) + (x << 3) + (c ^ 48), c = getchar();
            return (e |= c == EOF) ? 0 : ((f ? x = -x : 0), 1);
        }
        template <class _Tp>
        inline void write(_Tp x) {
            if (x < 0) putchar('-'), x = -x;
            if (x > 9) write(x / 10);
            putchar((x % 10) ^ 48);
        }
        template <typename T, typename... V>
        inline void read(T &t, V &...v) { read(t), read(v...); }
        template <typename T, typename... V>
        inline void write(T t, V... v) {
            write(t), putchar('␣'), write(v...);
        }
    }
}
using namespace fastIO::usr;
```

## 3 Watashi 代码库 (备用)

### 3.1 $O(n \log n) - O(1)$ RMQ

Listing 5: **rmq.cpp**

```
#include <algorithm> // copy
#include <climits>   // CHAR_BIT

using namespace std;

template <typename T>
struct RMQ {
    int n;
    vector<T> e;
    vector<vector<int>> rmq;

    static const int INT_BIT = sizeof(4) * CHAR_BIT;
    static inline int LG2(int i) { return INT_BIT − 1 − __builtin_clz(i); }
    static inline int BIN(int i) { return 1 << i; }

    int cmp(int l, int r) const {
        return e[l] <= e[r] ? l : r;
    }

    void init(int n, const T e[]) {
        this−>n = n;
        vector<T>(e, e + n).swap(this−>e);

        int m = 1;
        while (BIN(m) <= n) {
            ++m;
        }
        vector<vector<int>>(m, vector<int>(n)).swap(rmq);

        for (int i = 0; i < n; ++i) {
            rmq[0][i] = i;
        }
        for (int i = 0; BIN(i + 1) <= n; ++i) {
            for (int j = 0; j + BIN(i + 1) <= n; ++j) {
                rmq[i + 1][j] = cmp(rmq[i][j], rmq[i][j + BIN(i)]);
            }
        }
    }

    int index(int l, int r) const {
        int b = LG2(r − l);
        return cmp(rmq[b][l], rmq[b][r − (1 << b)]);
    }

    T value(int l, int r) const {
        return e[index(l, r)];
    }
};
```

### 3.2 $O(n \log n) - O(\log n)$ LCA

Listing 6: **lca.cpp**

```cpp
#include <algorithm>
#include <cstdio>
#include <vector>

using namespace std;

const int MAXM = 16;
const int MAXN = 1 << MAXM;

// LCA
struct LCA {
    vector<int> e[MAXN];
    int d[MAXN], p[MAXN][MAXM];

    void dfs_(int v, int f) {
        p[v][0] = f;
        for (int i = 1; i < MAXM; ++i) {
            p[v][i] = p[p[v][i - 1]][i - 1];
        }
        for (int i = 0; i < (int)e[v].size(); ++i) {
            int w = e[v][i];
            if (w != f) {
                d[w] = d[v] + 1;
                dfs_(w, v);
            }
        }
    }

    int up_(int v, int m) {
        for (int i = 0; i < MAXM; ++i) {
            if (m & (1 << i)) {
                v = p[v][i];
            }
        }
        return v;
    }

    int lca(int a, int b) {
        if (d[a] > d[b]) {
            swap(a, b);
        }
        b = up_(b, d[b] - d[a]);
        if (a == b) {
            return a;
        } else {
            for (int i = MAXM - 1; i >= 0; --i) {
                if (p[a][i] != p[b][i]) {
                    a = p[a][i];
                    b = p[b][i];
                }
            }
            return p[a][0];
        }
    }

    void init(int n) {
        for (int i = 0; i < n; ++i) {
            e[i].clear();
        }
    }

    void add(int a, int b) {
        e[a].push_back(b);
        e[b].push_back(a);
    }

    void build() {
        d[0] = 0;
        dfs_(0, 0);
    }
} lca;
```

## 3.3 树状数组

Listing 7: **bit.cpp**

```
#include <vector>

using namespace std;

template<typename T = int>
struct BIT {
  vector<T> a;

  void init(int n) {
    vector<T>(n + 1).swap(a);
  }

  void add(int i, T v) {
    for (int j = i + 1; j < (int)a.size(); j = (j | (j - 1)) + 1) {
      a[j] += v;
    }
  }

  // [0, i)
  T sum(int i) const {
    T ret = T();
    for (int j = i; j > 0; j = j & (j - 1)) {
      ret += a[j];
    }
    return ret;
  }

  T get(int i) const {
    return sum(i + 1) - sum(i);
  }

  void set(int i, T v) {
    add(i, v - get(i));
  }
};
```

## 3.4 并查集

Listing 8: **union-find.cpp**

```
#include <vector>

using namespace std;

struct DisjointSet {
    vector<int> p;

    void init(int n) {
        p.resize(n);
        for (int i = 0; i < n; ++i) {
            p[i] = i;
        }
    }

    int getp(int i) {
        return i == p[i] ? i : (p[i] = getp(p[i]));
    }

    bool setp(int i, int j) {
        i = getp(i);
        j = getp(j);
        p[i] = j;
        return i != j;
    }
};
```

## 3.5 轻重权树剖分

Listing 9: **chain-decomp.cpp**

```
#include <cstdio>
#include <vector>
#include <algorithm>

using namespace std;

const int MAXM = 16;
const int MAXN = 1 << MAXM;

// Heavy-Light Decomposition
struct TreeDecomposition {
  vector<int> e[MAXN], c[MAXN];
  int s[MAXN];    // subtree size
  int p[MAXN];    // parent id
  int r[MAXN];    // chain root id
  int t[MAXN];    // timestamp, index used in segtree
  int ts;

  void dfs_(int v, int f) {
    p[v] = f;
    s[v] = 1;
    for (int i = 0; i < (int)e[v].size(); ++i) {
      int w = e[v][i];
      if (w != f) {
        dfs_(w, v);
        s[v] += s[w];
      }
    }
  }

  void decomp_(int v, int f, int k) {
    t[v] = ts++;
    c[k].push_back(v);
    r[v] = k;

    int x = 0, y = -1;
    for (int i = 0; i < (int)e[v].size(); ++i) {
      int w = e[v][i];
      if (w != f) {
        if (s[w] > x) {
          x = s[w];
          y = w;
        }
      }
    }
    if (y != -1) {
      decomp_(y, v, k);
    }

    for (int i = 0; i < (int)e[v].size(); ++i) {
      int w = e[v][i];
      if (w != f && w != y) {
        decomp_(w, v, w);
      }
    }
  }

  void init(int n) {
    for (int i = 0; i < n; ++i) {
      e[i].clear();
    }
  }

  void add(int a, int b) {
    e[a].push_back(b);
    e[b].push_back(a);
  }

  void build() {  // !!
    ts = 0;
    dfs_(0, 0);
    decomp_(0, 0, 0);
  }
} hld;
```

## 3.6 强连通分量

Listing 10: **scc.cpp**

```
#include <algorithm>
#include <stack>
#include <vector>

using namespace std;

struct SCCTarjan {
    int n;
    vector<vector<int>> e;

    vector<int> id;
    vector<vector<int>> scc;

    void init(int n) {
        this->n = n;
        vector<vector<int>>(n).swap(e);
        id.resize(n);
        dfn.resize(n);
        low.resize(n);
    }

    void add(int a, int b) {
        e[a].push_back(b);
    }

    vector<int> dfn, low;
    int timestamp;
    stack<int> s;

    void dfs(int v) {
        dfn[v] = timestamp++;
        low[v] = dfn[v];
        s.push(v);
        for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
            if (dfn[*w] == -1) {
                dfs(*w);
                low[v] = min(low[v], low[*w]);
```

```cpp
            } else if (dfn[*w] != -2) {
                low[v] = min(low[v], dfn[*w]);
            }
        }

        if (low[v] == dfn[v]) {
            vector<int> t;
            do {
                int w = s.top();
                s.pop();
                id[w] = (int)scc.size();
                t.push_back(w);
                dfn[w] = -2;
            } while (t.back() != v);
            scc.push_back(t);
        }
    }

    int gao() {
        scc.clear();
        stack<int>().swap(s);
        timestamp = 0;

        fill(dfn.begin(), dfn.end(), -1);
        for (int i = 0; i < n; ++i) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return (int)scc.size();
    }
};
```

## 3.7 双连通分量

Listing 11: **bcc.cpp**

```cpp
#include <algorithm>
#include <stack>
#include <utility>
#include <vector>

using namespace std;

// TODO: cannot handle duplicate edges
struct Tarjan {
    int n;
    vector<vector<int>> e;

    vector<int> cut;
    vector<pair<int, int>> bridge;
    vector<vector<pair<int, int>>> bcc;

    void init(int n) {
        this->n = n;
        e.clear();
        e.resize(n);
        dfn.resize(n);
        low.resize(n);
    }

    void add(int a, int b) {
        // assert(find(e[a].begin(), e[a].end(), b) == e[a].end());
        e[a].push_back(b);
        e[b].push_back(a);
    }

    vector<int> dfn, low;
    int timestamp;
    stack<pair<int, int>> s;

    void dfs(int v, int p) {
        int part = p == -1 ? 0 : 1;
        dfn[v] = low[v] = timestamp++;
        for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
            pair<int, int> f = make_pair(min(v, *w), max(v, *w));
            if (dfn[*w] == -1) {
                s.push(f);
                dfs(*w, v);
                low[v] = min(low[v], low[*w]);
                if (dfn[v] <= low[*w]) {
                    // articulation point
                    if (++part == 2) {
                        cut.push_back(v);
                    }
                    // articulation edge
                    if (dfn[v] < low[*w]) {
                        bridge.push_back(f);
                    }
                    // biconnected component (2-vertex-connected)
                    vector<pair<int, int>> t;
                    do {
                        t.push_back(s.top());
                        s.pop();
                    } while (t.back() != f);
                    bcc.push_back(t);
                }
            } else if (*w != p && dfn[*w] < dfn[v]) {
                s.push(f);
                low[v] = min(low[v], dfn[*w]);
            }
        }
    }

    void gao() {
        cut.clear();
        bridge.clear();
        bcc.clear();

        timestamp = 0;
        stack<pair<int, int>>().swap(s);
        fill(dfn.begin(), dfn.end(), -1);

        for (int i = 0; i < n; ++i) {
            if (dfn[i] == -1) {
                dfs(i, -1);
            }
        }
    }
};

struct BridgeBlockTree {
    Tarjan<MAXN> bcc;
    DisjointSet<MAXN> ds;
    vector<int> e[MAXN];

    void init(int n) {
        bcc.init(n);
        ds.init(n);
    }

    void add(int a, int b) {
        bcc.add(a, b);
    }

    void gao() {
        bcc.gao();
        for (const auto &i : bcc.bcc) {
            if (i.size() > 1) {
                for (const auto &j : i) {
                    ds.setp(j.first, j.second);
                }
            }
        }
        for (const auto &i : bcc.bridge) {
            int a = ds.getp(i.first);
            int b = ds.getp(i.second);
            e[a].push_back(b);
            e[b].push_back(a);
        }
    }

    int id(int v) {
        return ds.getp(v);
    }
};
```

## 3.8 二分图匹配

Listing 12: **bimatch.cpp**

```
// maximum matchings in bipartite graphs
// maximum cardinality bipartite matching
// O(|V||E|), generally fast

#include <algorithm>
#include <string>
#include <vector>

using namespace std;

struct Hungarian {
    int nx, ny;
    vector<int> mx, my;
    vector<vector<int>> e;

    void init(int nx, int ny) {
        this->nx = nx;
        this->ny = ny;
        mx.resize(nx);
        my.resize(ny);
        e.clear();
        e.resize(nx);
        mark.resize(nx);
    }

    void add(int a, int b) {
        e[a].push_back(b);
    }

    // vector<bool> is evil!!!
    basic_string<bool> mark;

    bool augment(int i) {
        if (!mark[i]) {
            mark[i] = true;
            for (vector<int>::const_iterator j = e[i].begin(); j != e[i].end(); ++j) {
                if (my[*j] == -1 || augment(my[*j])) {
```

```
                mx[i] = *j;
                my[*j] = i;
                return true;
            }
        }
    }
    return false;
}

int gao() {
    int ret = 0;
    fill(mx.begin(), mx.end(), -1);
    fill(my.begin(), my.end(), -1);
    for (int i = 0; i < nx; ++i) {
        fill(mark.begin(), mark.end(), false);
        if (augment(i)) {
            ++ret;
        }
    }
    return ret;
}
};
```

## 3.9 最小费用最大流

Listing 13: **flow.cpp**

```
#include <algorithm>
#include <cstdio>
#include <limits>
#include <queue>
#include <vector>

using namespace std;

template <int MAXN, typename T = int, typename S = T>
struct MinCostMaxFlow {
    struct NegativeCostCircuitExistsException {
    };

    struct Edge {
        int v;
        T c;
        S w;
        int b;
        Edge(int v, T c, S w, int b) : v(v), c(c), w(w), b(b) {}
    };

    int n, source, sink;
    vector<Edge> e[MAXN];

    void init(int n, int source, int sink) {
        this->n = n;
        this->source = source;
        this->sink = sink;
        for (int i = 0; i < n; ++i) {
            e[i].clear();
        }
    }

    void addEdge(int a, int b, T c, S w) {
        e[a].push_back(Edge(b, c, w, e[b].size()));
        e[b].push_back(Edge(a, 0, -w, e[a].size() - 1)); // TODO
    }

    bool mark[MAXN];
    T maxc[MAXN];
    S minw[MAXN];
    int dist[MAXN];
    Edge *prev[MAXN];

    bool _spfa() {
        queue<int> q;
        fill(mark, mark + n, false);
        fill(maxc, maxc + n, 0);
        fill(minw, minw + n, numeric_limits<S>::max());
        fill(dist, dist + n, 0);
        fill(prev, prev + n, (Edge *)NULL);
        mark[source] = true;
        maxc[source] = numeric_limits<S>::max();
        minw[source] = 0;

        q.push(source);
        while (!q.empty()) {
            int cur = q.front();
            mark[cur] = false;
            q.pop();
            for (typename vector<Edge>::iterator it = e[cur].begin(); it != e[cur].end(); ++it) {
                T c = min(maxc[cur], it->c);
                if (c == 0) {
                    continue;
                }

                int v = it->v;
                S w = minw[cur] + it->w;
                if (minw[v] > w || (minw[v] == w && maxc[v] < c)) { // TODO
                    maxc[v] = c;
                    minw[v] = w;
                    dist[v] = dist[cur] + 1;
                    if (dist[v] >= n) {
                        return false;
                    }
                    prev[v] = &*it;
                    if (!mark[v]) {
                        mark[v] = true;
                        q.push(v);
                    }
                }
            }
        }
        return true;
    }

    pair<T, S> gao() {
        T sumc = 0;
        S sumw = 0;
        while (true) {
            if (!_spfa()) {
                throw NegativeCostCircuitExistsException();
            } else if (maxc[sink] == 0) {
                break;
            } else {
                T c = maxc[sink];
                sumc += c;
                sumw += c * minw[sink];

                int cur = sink;
                while (cur != source) {
                    Edge *e1 = prev[cur];
                    e1->c -= c;
                    Edge *e2 = &e[e1->v][e1->b];
                    e2->c += c;
                    cur = e2->v;
                }
            }
        }
        return make_pair(sumc, sumw);
    }
};
```

## 3.10 AhoCorasick 自动机

Listing 14: **ac-automata.cpp**

```
#include <algorithm>
#include <queue>

using namespace std;

struct AhoCorasick {
    static const int NONE = 0;
    static const int MAXN = 1024;
    static const int CHARSET = 26;

    int end;
    int tag[MAXN];
    int fail[MAXN];
    int trie[MAXN][CHARSET];

    void init() {
        tag[0] = NONE;
        fill(trie[0], trie[0] + CHARSET, -1);
        end = 1;
    }

    int add(int m, const int *s) {
        int p = 0;
        for (int i = 0; i < m; ++i) {
            if (trie[p][*s] == -1) {
                tag[end] = NONE;
                fill(trie[end], trie[end] + CHARSET, -1);
                trie[p][*s] = end++;
            }
            p = trie[p][*s];
            ++s;
        }
        return p;
    }

    void build(void) { // !!
        queue<int> bfs;
        fail[0] = 0;
        for (int i = 0; i < CHARSET; ++i) {
            if (trie[0][i] != -1) {
                fail[trie[0][i]] = 0;
                bfs.push(trie[0][i]);
            } else {
                trie[0][i] = 0;
            }
        }
        while (!bfs.empty()) {
            int p = bfs.front();
            tag[p] |= tag[fail[p]];
            bfs.pop();
            for (int i = 0; i < CHARSET; ++i) {
                if (trie[p][i] != -1) {
                    fail[trie[p][i]] = trie[fail[p]][i];
                    bfs.push(trie[p][i]);
                } else {
                    trie[p][i] = trie[fail[p]][i];
                }
            }
        }
    }
} ac;
```

## 3.11 后缀数组

Listing 15: **sa.cpp**

```
#include <algorithm>
#include <utility>
#include <vector>
using namespace std;

struct SuffixArray {
    vector<int> sa, rank, height;

    template <typename T>
    void init(int n, const T a[]) {
        sa.resize(n);
        rank.resize(n);

        vector<pair<T, int>> assoc(n);
        for (int i = 0; i < n; ++i) {
            assoc[i] = make_pair(a[i], i);
        }
        sort(assoc.begin(), assoc.end());
        for (int i = 0; i < n; ++i) {
            sa[i] = assoc[i].second;
            if (i == 0 || assoc[i].first != assoc[i - 1].first) {
                rank[sa[i]] = i;
            } else {
                rank[sa[i]] = rank[sa[i - 1]];
            }
        }

        vector<int> tmp(n), cnt(n);
        vector<pair<int, int>> suffix(n);
        for (int m = 1; m < n; m <<= 1) {
            // snd
            for (int i = 0; i < m; ++i) {
                tmp[i] = n - m + i;
            }
            for (int i = 0, j = m; i < n; ++i) {
                if (sa[i] >= m) {
                    tmp[j++] = sa[i] - m;
                }
            }
            // fst
            fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i) {
                ++cnt[rank[i]];
            }
            partial_sum(cnt.begin(), cnt.end(), cnt.begin());
            for (int i = n - 1; i >= 0; --i) {
                sa[--cnt[rank[tmp[i]]]] = tmp[i];
            }
            //
            for (int i = 0; i < n; ++i) {
                suffix[i] = make_pair(rank[i], i + m < n ? rank[i + m] : numeric_limits<int>::min());
            }
            for (int i = 0; i < n; ++i) {
                if (i == 0 || suffix[sa[i]] != suffix[sa[i - 1]]) {
```

```
                rank[sa[i]] = i;
            } else {
                rank[sa[i]] = rank[sa[i - 1]];
            }
        }
    }

    height.resize(n);
    for (int i = 0, z = 0; i < n; ++i) {
        if (rank[i] == 0) {
            height[0] = z = 0;
        } else {
            int x = i, y = sa[rank[i] - 1];
            z = max(0, z - 1);
            while (x + z < n && y + z < n && a[x + z] == a[y + z]) {
                ++z;
            }
            height[rank[i]] = z;
        }
    }
}
};
```

## 3.12 LU 分解

Listing 16: **lu.cpp**

```
const int MAXN = 128;
const double EPS = 1e-10;

void LU(int n, double a[MAXN][MAXN], int r[MAXN], int c[MAXN]) {
    for (int i = 0; i < n; ++i) {
        r[i] = c[i] = i;
    }
    for (int k = 0; k < n; ++k) {
        int ii = k, jj = k;
        for (int i = k; i < n; ++i) {
            for (int j = k; j < n; ++j) {
                if (fabs(a[i][j]) > fabs(a[ii][jj])) {
                    ii = i;
                    jj = j;
                }
            }
        }
        swap(r[k], r[ii]);
        swap(c[k], c[jj]);
        for (int i = 0; i < n; ++i) {
            swap(a[i][k], a[i][jj]);
        }
        for (int j = 0; j < n; ++j) {
            swap(a[k][j], a[ii][j]);
        }
        if (fabs(a[k][k]) < EPS) {
            continue;
        }
        for (int i = k + 1; i < n; ++i) {
            a[i][k] = a[i][k] / a[k][k];
            for (int j = k + 1; j < n; ++j) {
                a[i][j] -= a[i][k] * a[k][j];
            }
        }
    }
}

void solve(int n, double a[MAXN][MAXN], int r[MAXN], int c[MAXN], double b[MAXN]) {
    static double x[MAXN];
    for (int i = 0; i < n; ++i) {
        x[i] = b[r[i]];
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            x[i] -= a[i][j] * x[j];
        }
    }
    for (int i = n - 1; i >= 0; --i) {
        for (int j = n - 1; j > i; --j) {
            x[i] -= a[i][j] * x[j];
        }
        if (fabs(a[i][i]) >= EPS) {
            x[i] /= a[i][i];
        } // else assert(fabs(x[i]) < EPS);
    }
    for (int i = 0; i < n; ++i) {
        b[c[i]] = x[i];
    }
}

// LU(n - 1, a, r, c);
// solve(n - 1, a, r, c, b);
```