Alfred
代码模版库

目录

1 比赛配置 and 奇技淫巧

1.1 多组数据代码模板

Listing 1: **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;
}
```

1.2 快读快写

Listing 2: **fast-io.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;
```

1.3 关闭流与 C 风格输入输出的同步

Listing 3: **io-sync-off.cpp**

```cpp
#include <iostream>

inline void optimizeIO(void) {
    std::ios::sync_with_stdio(false);
    std::cin.tie(NULL), std::cout.tie(NULL);
}
```

1.4 .clang-format

Listing 4: **.clang-format**

```
BasedOnStyle: LLVM
AlignAfterOpenBracket: BlockIndent
# AlignConsecutiveAssignments: Consecutive
AlignArrayOfStructures: Right
UseTab: Never
IndentWidth: 4
TabWidth: 4
BreakBeforeBraces: Attach
AllowShortIfStatementsOnASingleLine: AllIfsAndElse
AllowShortLoopsOnASingleLine: true
AllowShortBlocksOnASingleLine: true
IndentCaseLabels: true
ColumnLimit: 0
AccessModifierOffset: -4
NamespaceIndentation: All
FixNamespaceComments: false
AllowShortCaseLabelsOnASingleLine: true
AlwaysBreakTemplateDeclarations: MultiLine
BinPackParameters: true
BraceWrapping:
  AfterCaseLabel: true
  AfterClass: true
AlignConsecutiveMacros: AcrossEmptyLinesAndComments
AlignTrailingComments: Always
```

1.5 debug.h

Listing 5: **debug.h**

```
/**
 * @file        debug.h
 * @author      Dr.Alfred (abonlinejudge@163.com)
 * @brief       Local Debug Printer
 * @version     1.0
 * @date        2023-12-30
 *
 * @copyright   Copyright (c) 2019-now <Rhodes Island Inc.>
 *
 */

#include <bits/stdc++.h>

using std::cerr;
using std::pair;
using std::string;

const long long dbg_inf = 9e18 + 19260817;

void __print(int x) { cerr << x; }
void __print(long x) { cerr << x; }
void __print(long long x) {
    if (x != dbg_inf) {
        cerr << x;
    } else {
        cerr << "inf";
    }
}
void __print(unsigned x) { cerr << x; }
void __print(unsigned long x) { cerr << x; }
void __print(unsigned long long x) { cerr << x; }
void __print(float x) { cerr << x; }
void __print(double x) { cerr << x; }
```

```
34 void __print(long double x) { cerr << x; }
35 void __print(char x) { cerr << '\'' << x << '\''; }
36 void __print(const char *x) { cerr << '\"' << x << '\"'; }
37 void __print(const string &x) { cerr << '\"' << x << '\"'; }
38 void __print(bool x) { cerr << (x ? "true" : "false"); }
39 void __print(__int128_t x) {
40     if (x < 0) cerr << '-', x = -x;
41     if (x > 9) __print(x / 10);
42     cerr << char((x % 10) ^ 48);
43 }
44 void dbgEndl(void) { cerr << '\n'; }
45
46 template <typename T, typename V>
47 void __print(const pair<T, V> &x) {
48     cerr << '{', __print(x.first), cerr << ", ", __print(x.second), cerr << '}';
49 }
50 template <typename T>
51 void __print(const T &x) {
52     int f = 0;
53     cerr << '{';
54     for (auto i : x) cerr << (f++ ? ", " : ""), __print(i);
55     cerr << "}";
56 }
57 void _print() { cerr << "]\n"; }
58 template <typename T, typename... V>
59 void _print(T t, V... v) {
60     __print(t);
61     if (sizeof...(v)) cerr << ", ";
62     _print(v...);
63 }
64 #ifdef DEBUG
65 // To customize a struct/class to print, just define the __print function.
66
67 #ifndef NO_DBG_COLOR
68 #define dbg(x...)                                                          \
69     cerr << "\e[91m" << __func__ << ":" << __LINE__ << " [" << #x << "] = ["; \
70     _print(x);                                                             \
71     cerr << "\e[39m";
72
73 #define short_dbg(x...) \
74     cerr << "\e[91m[["; \
75     _print(x);         \
76     cerr << "\e[39m";
77 #else
78 #define dbg(x...)                                          \
79     cerr << __func__ << ":" << __LINE__ << " [" << #x << "] = ["; \
80     _print(x);
81 #define short_dbg(x...) \
82     cerr << "[";        \
83     _print(x);
84 #endif // !NO_DBG_COLOR
85
86 #else
87 #define dbg(x...)
88 #endif
```

2 数据结构

2.1 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 6: **chtholly.cpp**

```
1 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;
    }
};
```

2.2 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 7: **fenwick.cpp**

```
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; }
    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) {}
```

```
};
```

2.3 静态可重区间信息（支持 RMQ）

基于 ST 表，支持静态数组可重区间信息的数据结构。

Listing 8: **sparse-table.cpp**

```cpp
template <class T>
struct MaxInfo {
    T val;
    MaxInfo() { val = T(); }
    template <class InitT>
    MaxInfo(InitT x) { val = x; }
    MaxInfo operator+(MaxInfo &x) {
        return {std::max(val, x.val)};
    }
};
template <class T>
struct MinInfo {
    T val;
    MinInfo() { val = T(); }
    template <class InitT>
    MinInfo(InitT x) { val = x; }
    MinInfo operator+(MinInfo &x) {
        return {std::min(val, x.val)};
    }
};
template <class T>
struct GcdInfo {
    T val;
    GcdInfo() { val = T(); }
    template <class InitT>
    GcdInfo(InitT x) { val = x; }
    GcdInfo operator+(GcdInfo &x) {
        return {std::gcd(val, x.val)};
    }
};
template <class T>
struct SparseTable {
private:
    int n;
    std::vector<std::vector<T>> ST;

public:
    SparseTable() {}
    SparseTable(int N) : n(N), ST(N, std::vector<T>(__lg(N) + 1)) {}
    template <class InitT>
    SparseTable(std::vector<InitT> init) : SparseTable(init.size()) {
        for (int i = 0; i < n; i++) ST[i][0] = T(init[i]);
        for (int i = 1; (1 << i) <= n; i++) {
            for (int j = 0; j + (1 << i) - 1 < n; j++) {
                ST[j][i] = ST[j][i - 1] + ST[j + (1 << (i - 1))][i - 1];
            }
        }
    }
    inline T query(int l, int r) { // 0 based
        int w = std::__lg(r - l + 1);
        return ST[l][w] + ST[r - (1 << w) + 1][w];
    }
};
```

2.4 PBDS 大常数平衡树

GNU PBDS 提供的大常数基于 rb-tree 的平衡树。

Listing 9: **pbds-balance-tree.cpp**

```cpp
#include <bits/extc++.h>
#include <bits/stdc++.h>

using namespace std;
using namespace __gnu_pbds;

// TreeTag can also be __gnu_pbds::splay_tree_tag
template <class T, class Cmp, class TreeTag = rb_tree_tag>
using BalanceTree = tree<T, null_type, Cmp, TreeTag, tree_order_statistics_node_update>;
```

2.5 离散化容器

Listing 10: **discretization.cpp**

```cpp
template <class T>
struct Mess {
    std::vector<T> v;
    bool initialized = false;
    inline T origin(int idx) { return v[idx - 1]; }
    inline void insert(T x) { v.push_back(x); }
    template <typename T, typename... V>
    inline void insert(T x, V... v) { insert(x), insert(v...); }
    inline void init(void) {
        sort(v.begin(), v.end()), initialized = true;
        v.erase(unique(v.begin(), v.end()), v.end());
    }
    inline int query(T x) {
        if (!initialized) init();
        return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
    }
};
```

2.6 并查集

Listing 11: **dsu.cpp**

```cpp
struct DSU {
    std::vector<int> fa;
    DSU(int n) : fa(n + 1) {
        iota(fa.begin(), fa.end(), 0);
    }
    inline int find(int x) {
        return fa[x] == x ? x : fa[x] = find(fa[x]);
    }
    inline void merge(int x, int y) {
        int fx = find(x), fy = find(y);
        if (fx != fy) fa[fx] = fy;
    }
};
```

3 数学（数论）算法

3.1 带模整数类

Listing 12: **mod-int.cpp**

```cpp
template <int mod>
inline unsigned down(unsigned x) { return x >= mod ? x - mod : x; }
template <int mod>
```

```
4  struct ModInt {
5      unsigned int x;
6      ModInt() = default;
7      ModInt(unsigned int x) : x(x) {}
8      friend istream &operator>>(istream &in, ModInt &a) { return in >> a.x; }
9      friend ostream &operator<<(ostream &out, ModInt a) { return out << a.x; }
10     friend ModInt operator+(ModInt a, ModInt b) { return down<mod>(a.x + b.x); }
11     friend ModInt operator-(ModInt a, ModInt b) { return down<mod>(a.x - b.x + mod); }
12     friend ModInt operator*(ModInt a, ModInt b) { return 1ULL * a.x * b.x % mod; }
13     friend ModInt operator/(ModInt a, ModInt b) { return a * ~b; }
14     friend ModInt operator^(ModInt a, int b) {
15         ModInt ans = 1;
16         for (; b; b >>= 1, a *= a)
17             if (b & 1) ans *= a;
18         return ans;
19     }
20     friend ModInt operator~(ModInt a) { return a ^ (mod - 2); }
21     friend ModInt operator-(ModInt a) { return down<mod>(mod - a.x); }
22     friend ModInt &operator+=(ModInt &a, ModInt b) { return a = a + b; }
23     friend ModInt &operator-=(ModInt &a, ModInt b) { return a = a - b; }
24     friend ModInt &operator*=(ModInt &a, ModInt b) { return a = a * b; }
25     friend ModInt &operator/=(ModInt &a, ModInt b) { return a = a / b; }
26     friend ModInt &operator^=(ModInt &a, int b) { return a = a ^ b; }
27     friend ModInt &operator++(ModInt &a) { return a += 1; }
28     friend ModInt operator++(ModInt &a, int) {
29         ModInt x = a;
30         a += 1;
31         return x;
32     }
33     friend ModInt &operator--(ModInt &a) { return a -= 1; }
34     friend ModInt operator--(ModInt &a, int) {
35         ModInt x = a;
36         a -= 1;
37         return x;
38     }
39     friend bool operator==(ModInt a, ModInt b) { return a.x == b.x; }
40     friend bool operator!=(ModInt a, ModInt b) { return !(a == b); }
41 };
42 const int p = 1e9 + 7;
43 using mint = ModInt<p>;
```

4 字符串算法

4.1 字符串哈希

Listing 13: **hashed-string.cpp**

```
1  using i64 = long long;
2  using i128 = __int128;
3  class HashedString {
4  private:
5      // change M and B if you want
6      static const i64 M = (1LL << 61) - 1;
7      static const i64 B;
8      // pow[i] contains B^i % M
9      static std::vector<i64> pow;
10     // p_hash[i] is the hash of the first i characters of the given string
11     std::vector<i64> r_hash, p_hash;
12     i128 mul(i64 a, i64 b) { return (i128)a * b; }
13     i64 mod_mul(i64 a, i64 b) { return mul(a, b) % M; }
14
15 public:
16     HashedString(const string &s) : r_hash(s.size() + 1), p_hash(s.size() + 1) {
17         while (pow.size() < s.size()) { pow.push_back(mod_mul(pow.back(), B)); }
18         p_hash[0] = 0;
```

```
19          r_hash[0] = 0;
20          for (size_t i = 0; i < s.size(); i++) {
21              p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M; // 1-based
22          }
23          i64 sz = s.size();
24          for (int i = sz - 1, j = 0; i >= 0; i--, j++) {
25              r_hash[j + 1] = (mul(r_hash[j], B) + s[i]) % M;
26          }
27      }
28      i64 getHash(int start, int end) { // 0 based
29          i64 raw_val = p_hash[end + 1] - mod_mul(p_hash[start], pow[end - start + 1]);
30          return (raw_val + M) % M;
31      }
32      i64 getRHash(int start, int end) { // 0 based
33          i64 raw_val = r_hash[end + 1] - mod_mul(r_hash[start], pow[end - start + 1]);
34          return (raw_val + M) % M;
35      }
36  };
37  std::vector<i64> HashedString::pow = {1};
38  mt19937 rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
39  const i64 HashedString::B = uniform_int_distribution<i64>(0, M - 1)(rng);
```

5 jiangly 代码库 (备用，侵权请提出 issue)

5.1 int128 输出流自定义

Listing 14: **others/i128-stream.cpp**

```
1   #include <iostream>
2
3   using i128 = __int128;
4
5   std::istream &operator>>(std::istream is, i128 &n) {
6       std::string s;
7       is >> s;
8       for (auto c : s) {
9           n = n * 10 + (c - '0');
10      }
11      return is;
12  }
13
14  std::ostream &operator<<(std::ostream &os, i128 n) {
15      std::string s;
16      while (n) {
17          s += '0' + n % 10;
18          n /= 10;
19      }
20      std::reverse(s.begin(), s.end());
21      return os << s;
22  }
```

5.2 常用数学运算库函数及 gcd 重载

Listing 15: **others/clf.cpp**

```
1   using i64 = long long;
2   using i128 = __int128;
3
4   i64 ceilDiv(i64 n, i64 m) {
5       if (n >= 0) {
6           return (n + m - 1) / m;
7       } else {
8           return n / m;
9       }
```

```
}

i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
    } else {
        return (n - m + 1) / m;
    }
}

template <class T>
void chmax(T &a, T b) {
    if (a < b) a = b;
}

i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}
```

5.3 强连通分量缩点 (SCC)

Listing 16: **graph/scc.cpp**

```
#include <vector>

struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

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

    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
```

```
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

5.4 割边与割边缩点 (EBCC)

Listing 17: **graph/ebcc.cpp**

```
#include <set>
#include <vector>

std::set<std::pair<int, int>> E;

struct EBCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    EBCC() {}
    EBCC(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }

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

    void dfs(int x, int p) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);

        for (auto y : adj[x]) {
            if (y == p) {
                continue;
            }
            if (dfn[y] == -1) {
                E.emplace(x, y);
                dfs(y, x);
```

```
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
                E.emplace(x, y);
                low[x] = std::min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
            cnt++;
        }
    }

    std::vector<int> work() {
        dfs(0, -1);
        return bel;
    }

    struct Graph {
        int n;
        std::vector<std::pair<int, int>> edges;
        std::vector<int> siz;
        std::vector<int> cnte;
    };
    Graph compress() {
        Graph g;
        g.n = cnt;
        g.siz.resize(cnt);
        g.cnte.resize(cnt);
        for (int i = 0; i < n; i++) {
            g.siz[bel[i]]++;
            for (auto j : adj[i]) {
                if (bel[i] < bel[j]) {
                    g.edges.emplace_back(bel[i], bel[j]);
                } else if (i < j) {
                    g.cnte[bel[i]]++;
                }
            }
        }
        return g;
    }
};
```

5.5 二分图最大权匹配 (MaxAssignment，基于 KM)

Listing 18: **graph/bigraph-weight-match.cpp**

```
#include <queue>
#include <vector>

template <class T>
struct MaxAssignment {
public:
    T solve(int nx, int ny, std::vector<std::vector<T>> a) {
        assert(0 <= nx && nx <= ny);
        assert(int(a.size()) == nx);
        for (int i = 0; i < nx; ++i) {
            assert(int(a[i].size()) == ny);
            for (auto x : a[i])
                assert(x >= 0);
        }
```

```
15
16          auto update = [&](int x) {
17              for (int y = 0; y < ny; ++y) {
18                  if (lx[x] + ly[y] - a[x][y] < slack[y]) {
19                      slack[y] = lx[x] + ly[y] - a[x][y];
20                      slackx[y] = x;
21                  }
22              }
23          };
24
25          costs.resize(nx + 1);
26          costs[0] = 0;
27          lx.assign(nx, std::numeric_limits<T>::max());
28          ly.assign(ny, 0);
29          xy.assign(nx, -1);
30          yx.assign(ny, -1);
31          slackx.resize(ny);
32          for (int cur = 0; cur < nx; ++cur) {
33              std::queue<int> que;
34              visx.assign(nx, false);
35              visy.assign(ny, false);
36              slack.assign(ny, std::numeric_limits<T>::max());
37              p.assign(nx, -1);
38
39              for (int x = 0; x < nx; ++x) {
40                  if (xy[x] == -1) {
41                      que.push(x);
42                      visx[x] = true;
43                      update(x);
44                  }
45              }
46
47              int ex, ey;
48              bool found = false;
49              while (!found) {
50                  while (!que.empty() && !found) {
51                      auto x = que.front();
52                      que.pop();
53                      for (int y = 0; y < ny; ++y) {
54                          if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
55                              if (yx[y] == -1) {
56                                  ex = x;
57                                  ey = y;
58                                  found = true;
59                                  break;
60                              }
61                              que.push(yx[y]);
62                              p[yx[y]] = x;
63                              visy[y] = visx[yx[y]] = true;
64                              update(yx[y]);
65                          }
66                      }
67                  }
68                  if (found)
69                      break;
70
71                  T delta = std::numeric_limits<T>::max();
72                  for (int y = 0; y < ny; ++y)
73                      if (!visy[y])
74                          delta = std::min(delta, slack[y]);
75                  for (int x = 0; x < nx; ++x)
76                      if (visx[x])
77                          lx[x] -= delta;
78                  for (int y = 0; y < ny; ++y) {
79                      if (visy[y]) {
80                          ly[y] += delta;
```

```
                    } else {
                        slack[y] -= delta;
                    }
                }
                for (int y = 0; y < ny; ++y) {
                    if (!visy[y] && slack[y] == 0) {
                        if (yx[y] == -1) {
                            ex = slackx[y];
                            ey = y;
                            found = true;
                            break;
                        }
                        que.push(yx[y]);
                        p[yx[y]] = slackx[y];
                        visy[y] = visx[yx[y]] = true;
                        update(yx[y]);
                    }
                }
            }

            costs[cur + 1] = costs[cur];
            for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
                costs[cur + 1] += a[x][y];
                if (xy[x] != -1)
                    costs[cur + 1] -= a[x][xy[x]];
                ty = xy[x];
                xy[x] = y;
                yx[y] = x;
            }
        }
        return costs[nx];
    }
    std::vector<int> assignment() {
        return xy;
    }
    std::pair<std::vector<T>, std::vector<T>> labels() {
        return std::make_pair(lx, ly);
    }
    std::vector<T> weights() {
        return costs;
    }

private:
    std::vector<T> lx, ly, slack, costs;
    std::vector<int> xy, yx, p, slackx;
    std::vector<bool> visx, visy;
};
```

5.6 一般图最大匹配 (Graph，带花树算法)

Listing 19: **graph/general-match.cpp**

```
#include <queue>
#include <vector>

struct Graph {
    int n;
    std::vector<std::vector<int>> e;
    Graph(int n) : n(n), e(n) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    std::vector<int> findMatching() {
        std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);

```

```
15          // disjoint set union
16          auto find = [&](int u) {
17              while (f[u] != u)
18                  u = f[u] = f[f[u]];
19              return u;
20          };
21
22          auto lca = [&](int u, int v) {
23              u = find(u);
24              v = find(v);
25              while (u != v) {
26                  if (dep[u] < dep[v])
27                      std::swap(u, v);
28                  u = find(link[match[u]]);
29              }
30              return u;
31          };
32
33          std::queue<int> que;
34          auto blossom = [&](int u, int v, int p) {
35              while (find(u) != p) {
36                  link[u] = v;
37                  v = match[u];
38                  if (vis[v] == 0) {
39                      vis[v] = 1;
40                      que.push(v);
41                  }
42                  f[u] = f[v] = p;
43                  u = link[v];
44              }
45          };
46
47          // find an augmenting path starting from u and augment (if exist)
48          auto augment = [&](int u) {
49              while (!que.empty())
50                  que.pop();
51
52              std::iota(f.begin(), f.end(), 0);
53
54              // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
55              std::fill(vis.begin(), vis.end(), -1);
56
57              que.push(u);
58              vis[u] = 1;
59              dep[u] = 0;
60
61              while (!que.empty()) {
62                  int u = que.front();
63                  que.pop();
64                  for (auto v : e[u]) {
65                      if (vis[v] == -1) {
66
67                          vis[v] = 0;
68                          link[v] = u;
69                          dep[v] = dep[u] + 1;
70
71                          // found an augmenting path
72                          if (match[v] == -1) {
73                              for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
74                                  temp = match[y];
75                                  match[x] = y;
76                                  match[y] = x;
77                              }
78                              return;
79                          }
80
```

```
                        vis[match[v]] = 1;
                        dep[match[v]] = dep[u] + 2;
                        que.push(match[v]);

                    } else if (vis[v] == 1 && find(v) != find(u)) {
                        // found a blossom
                        int p = lca(u, v);
                        blossom(u, v, p);
                        blossom(v, u, p);
                    }
                }
            }
        };

        // find a maximal matching greedily (decrease constant)
        auto greedy = [&]() {
            for (int u = 0; u < n; ++u) {
                if (match[u] != -1)
                    continue;
                for (auto v : e[u]) {
                    if (match[v] == -1) {
                        match[u] = v;
                        match[v] = u;
                        break;
                    }
                }
            }
        };

        greedy();

        for (int u = 0; u < n; ++u)
            if (match[u] == -1)
                augment(u);

        return match;
    }
};
```

5.7 2-SAT

Listing 20: **graph/2-sat.cpp**

```
#include <vector>

struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
```

```
                    low[u] = std::min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i)
            if (dfn[i] == -1) tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    std::vector<bool> answer() { return ans; }
};
```

6 Watashi 代码库 (备用)

6.1 $O(n \log n) - O(1)$ RMQ

Listing 21: **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)];
    }
};
```

6.2 $O(n\log n)-O(\log n)$ LCA

Listing 22: **lca.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;
```

6.3 树状数组

Listing 23: **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));
  }
};
```

6.4 并查集

Listing 24: **union-find.cpp**

```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;
    }
};
```

6.5 轻重权树剖分

Listing 25: **chain-decomp.cpp**

```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++;
```

```
33      c[k].push_back(v);
34      r[v] = k;
35
36      int x = 0, y = -1;
37      for (int i = 0; i < (int)e[v].size(); ++i) {
38        int w = e[v][i];
39        if (w != f) {
40          if (s[w] > x) {
41            x = s[w];
42            y = w;
43          }
44        }
45      }
46      if (y != -1) {
47        decomp_(y, v, k);
48      }
49
50      for (int i = 0; i < (int)e[v].size(); ++i) {
51        int w = e[v][i];
52        if (w != f && w != y) {
53          decomp_(w, v, w);
54        }
55      }
56    }
57
58    void init(int n) {
59      for (int i = 0; i < n; ++i) {
60        e[i].clear();
61      }
62    }
63
64    void add(int a, int b) {
65      e[a].push_back(b);
66      e[b].push_back(a);
67    }
68
69    void build() {  // !!
70      ts = 0;
71      dfs_(0, 0);
72      decomp_(0, 0, 0);
73    }
74  } hld;
```

6.6 强连通分量

Listing 26: **scc.cpp**

```
1   #include <algorithm>
2   #include <stack>
3   #include <vector>
4
5   using namespace std;
6
7   struct SCCTarjan {
8       int n;
9       vector<vector<int>> e;
10
11      vector<int> id;
12      vector<vector<int>> scc;
13
14      void init(int n) {
15          this->n = n;
16          vector<vector<int>>(n).swap(e);
17          id.resize(n);
18          dfn.resize(n);
19          low.resize(n);
```

```
20      }
21
22      void add(int a, int b) {
23          e[a].push_back(b);
24      }
25
26      vector<int> dfn, low;
27      int timestamp;
28      stack<int> s;
29
30      void dfs(int v) {
31          dfn[v] = timestamp++;
32          low[v] = dfn[v];
33          s.push(v);
34          for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
35              if (dfn[*w] == -1) {
36                  dfs(*w);
37                  low[v] = min(low[v], low[*w]);
38              } else if (dfn[*w] != -2) {
39                  low[v] = min(low[v], dfn[*w]);
40              }
41          }
42
43          if (low[v] == dfn[v]) {
44              vector<int> t;
45              do {
46                  int w = s.top();
47                  s.pop();
48                  id[w] = (int)scc.size();
49                  t.push_back(w);
50                  dfn[w] = -2;
51              } while (t.back() != v);
52              scc.push_back(t);
53          }
54      }
55
56      int gao() {
57          scc.clear();
58          stack<int>().swap(s);
59          timestamp = 0;
60
61          fill(dfn.begin(), dfn.end(), -1);
62          for (int i = 0; i < n; ++i) {
63              if (dfn[i] == -1) {
64                  dfs(i);
65              }
66          }
67          return (int)scc.size();
68      }
69  };
```

6.7 双连通分量

Listing 27: **bcc.cpp**

```
1   #include <algorithm>
2   #include <stack>
3   #include <utility>
4   #include <vector>
5
6   using namespace std;
7
8   // TODO: cannot handle duplicate edges
9   struct Tarjan {
10      int n;
11      vector<vector<int>> e;
```

```
12
13      vector<int> cut;
14      vector<pair<int, int>> bridge;
15      vector<vector<pair<int, int>>> bcc;
16
17      void init(int n) {
18          this->n = n;
19          e.clear();
20          e.resize(n);
21          dfn.resize(n);
22          low.resize(n);
23      }
24
25      void add(int a, int b) {
26          // assert(find(e[a].begin(), e[a].end(), b) == e[a].end());
27          e[a].push_back(b);
28          e[b].push_back(a);
29      }
30
31      vector<int> dfn, low;
32      int timestamp;
33      stack<pair<int, int>> s;
34
35      void dfs(int v, int p) {
36          int part = p == -1 ? 0 : 1;
37          dfn[v] = low[v] = timestamp++;
38          for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
39              pair<int, int> f = make_pair(min(v, *w), max(v, *w));
40              if (dfn[*w] == -1) {
41                  s.push(f);
42                  dfs(*w, v);
43                  low[v] = min(low[v], low[*w]);
44                  if (dfn[v] <= low[*w]) {
45                      // articulation point
46                      if (++part == 2) {
47                          cut.push_back(v);
48                      }
49                      // articulation edge
50                      if (dfn[v] < low[*w]) {
51                          bridge.push_back(f);
52                      }
53                      // biconnected component (2-vertex-connected)
54                      vector<pair<int, int>> t;
55                      do {
56                          t.push_back(s.top());
57                          s.pop();
58                      } while (t.back() != f);
59                      bcc.push_back(t);
60                  }
61              } else if (*w != p && dfn[*w] < dfn[v]) {
62                  s.push(f);
63                  low[v] = min(low[v], dfn[*w]);
64              }
65          }
66      }
67
68      void gao() {
69          cut.clear();
70          bridge.clear();
71          bcc.clear();
72
73          timestamp = 0;
74          stack<pair<int, int>>().swap(s);
75          fill(dfn.begin(), dfn.end(), -1);
76
77          for (int i = 0; i < n; ++i) {
```

```
78              if (dfn[i] == -1) {
79                  dfs(i, -1);
80              }
81          }
82      }
83  };
84
85  struct BridgeBlockTree {
86      Tarjan<MAXN> bcc;
87      DisjointSet<MAXN> ds;
88      vector<int> e[MAXN];
89
90      void init(int n) {
91          bcc.init(n);
92          ds.init(n);
93      }
94
95      void add(int a, int b) {
96          bcc.add(a, b);
97      }
98
99      void gao() {
100         bcc.gao();
101         for (const auto &i : bcc.bcc) {
102             if (i.size() > 1) {
103                 for (const auto &j : i) {
104                     ds.setp(j.first, j.second);
105                 }
106             }
107         }
108         for (const auto &i : bcc.bridge) {
109             int a = ds.getp(i.first);
110             int b = ds.getp(i.second);
111             e[a].push_back(b);
112             e[b].push_back(a);
113         }
114     }
115
116     int id(int v) {
117         return ds.getp(v);
118     }
119 };
```

6.8 二分图匹配

Listing 28: **bimatch.cpp**

```
1   // maximum matchings in bipartite graphs
2   // maximum cardinality bipartite matching
3   // O(|V||E|), generally fast
4
5   #include <algorithm>
6   #include <string>
7   #include <vector>
8
9   using namespace std;
10
11  struct Hungarian {
12      int nx, ny;
13      vector<int> mx, my;
14      vector<vector<int>> e;
15
16      void init(int nx, int ny) {
17          this->nx = nx;
18          this->ny = ny;
19          mx.resize(nx);
```

```
20          my.resize(ny);
21          e.clear();
22          e.resize(nx);
23          mark.resize(nx);
24      }
25
26      void add(int a, int b) {
27          e[a].push_back(b);
28      }
29
30      // vector<bool> is evil!!!
31      basic_string<bool> mark;
32
33      bool augment(int i) {
34          if (!mark[i]) {
35              mark[i] = true;
36              for (vector<int>::const_iterator j = e[i].begin(); j != e[i].end(); ++j) {
37                  if (my[*j] == -1 || augment(my[*j])) {
38                      mx[i] = *j;
39                      my[*j] = i;
40                      return true;
41                  }
42              }
43          }
44          return false;
45      }
46
47      int gao() {
48          int ret = 0;
49          fill(mx.begin(), mx.end(), -1);
50          fill(my.begin(), my.end(), -1);
51          for (int i = 0; i < nx; ++i) {
52              fill(mark.begin(), mark.end(), false);
53              if (augment(i)) {
54                  ++ret;
55              }
56          }
57          return ret;
58      }
59  };
```

6.9 最小费用最大流

Listing 29: **flow.cpp**

```
1   #include <algorithm>
2   #include <cstdio>
3   #include <limits>
4   #include <queue>
5   #include <vector>
6
7   using namespace std;
8
9   template <int MAXN, typename T = int, typename S = T>
10  struct MinCostMaxFlow {
11      struct NegativeCostCircuitExistsException {
12      };
13
14      struct Edge {
15          int v;
16          T c;
17          S w;
18          int b;
19          Edge(int v, T c, S w, int b) : v(v), c(c), w(w), b(b) {}
20      };
21
```

```
22      int n, source, sink;
23      vector<Edge> e[MAXN];
24
25      void init(int n, int source, int sink) {
26          this->n = n;
27          this->source = source;
28          this->sink = sink;
29          for (int i = 0; i < n; ++i) {
30              e[i].clear();
31          }
32      }
33
34      void addEdge(int a, int b, T c, S w) {
35          e[a].push_back(Edge(b, c, w, e[b].size()));
36          e[b].push_back(Edge(a, 0, -w, e[a].size() - 1)); // TODO
37      }
38
39      bool mark[MAXN];
40      T maxc[MAXN];
41      S minw[MAXN];
42      int dist[MAXN];
43      Edge *prev[MAXN];
44
45      bool _spfa() {
46          queue<int> q;
47          fill(mark, mark + n, false);
48          fill(maxc, maxc + n, 0);
49          fill(minw, minw + n, numeric_limits<S>::max());
50          fill(dist, dist + n, 0);
51          fill(prev, prev + n, (Edge *)NULL);
52          mark[source] = true;
53          maxc[source] = numeric_limits<S>::max();
54          minw[source] = 0;
55
56          q.push(source);
57          while (!q.empty()) {
58              int cur = q.front();
59              mark[cur] = false;
60              q.pop();
61              for (typename vector<Edge>::iterator it = e[cur].begin(); it != e[cur].end(); ++it) {
62                  T c = min(maxc[cur], it->c);
63                  if (c == 0) {
64                      continue;
65                  }
66
67                  int v = it->v;
68                  S w = minw[cur] + it->w;
69                  if (minw[v] > w || (minw[v] == w && maxc[v] < c)) { // TODO
70                      maxc[v] = c;
71                      minw[v] = w;
72                      dist[v] = dist[cur] + 1;
73                      if (dist[v] >= n) {
74                          return false;
75                      }
76                      prev[v] = &*it;
77                      if (!mark[v]) {
78                          mark[v] = true;
79                          q.push(v);
80                      }
81                  }
82              }
83          }
84          return true;
85      }
86
87      pair<T, S> gao() {
```

```
88          T sumc = 0;
89          S sumw = 0;
90          while (true) {
91              if (!_spfa()) {
92                  throw NegativeCostCircuitExistsException();
93              } else if (maxc[sink] == 0) {
94                  break;
95              } else {
96                  T c = maxc[sink];
97                  sumc += c;
98                  sumw += c * minw[sink];
99
100                 int cur = sink;
101                 while (cur != source) {
102                     Edge *e1 = prev[cur];
103                     e1->c -= c;
104                     Edge *e2 = &e[e1->v][e1->b];
105                     e2->c += c;
106                     cur = e2->v;
107                 }
108             }
109         }
110         return make_pair(sumc, sumw);
111     }
112 };
```

6.10 AhoCorasick 自动机

Listing 30: **ac-automata.cpp**

```
1  #include <algorithm>
2  #include <queue>
3
4  using namespace std;
5
6  struct AhoCorasick {
7      static const int NONE = 0;
8      static const int MAXN = 1024;
9      static const int CHARSET = 26;
10
11     int end;
12     int tag[MAXN];
13     int fail[MAXN];
14     int trie[MAXN][CHARSET];
15
16     void init() {
17         tag[0] = NONE;
18         fill(trie[0], trie[0] + CHARSET, -1);
19         end = 1;
20     }
21
22     int add(int m, const int *s) {
23         int p = 0;
24         for (int i = 0; i < m; ++i) {
25             if (trie[p][*s] == -1) {
26                 tag[end] = NONE;
27                 fill(trie[end], trie[end] + CHARSET, -1);
28                 trie[p][*s] = end++;
29             }
30             p = trie[p][*s];
31             ++s;
32         }
33         return p;
34     }
35
36     void build(void) { // !!
```

```
37          queue<int> bfs;
38          fail[0] = 0;
39          for (int i = 0; i < CHARSET; ++i) {
40              if (trie[0][i] != -1) {
41                  fail[trie[0][i]] = 0;
42                  bfs.push(trie[0][i]);
43              } else {
44                  trie[0][i] = 0;
45              }
46          }
47          while (!bfs.empty()) {
48              int p = bfs.front();
49              tag[p] |= tag[fail[p]];
50              bfs.pop();
51              for (int i = 0; i < CHARSET; ++i) {
52                  if (trie[p][i] != -1) {
53                      fail[trie[p][i]] = trie[fail[p]][i];
54                      bfs.push(trie[p][i]);
55                  } else {
56                      trie[p][i] = trie[fail[p]][i];
57                  }
58              }
59          }
60      }
61  } ac;
```

6.11 后缀数组

Listing 31: **sa.cpp**

```
1   #include <algorithm>
2   #include <utility>
3   #include <vector>
4   using namespace std;
5
6   struct SuffixArray {
7       vector<int> sa, rank, height;
8
9       template <typename T>
10      void init(int n, const T a[]) {
11          sa.resize(n);
12          rank.resize(n);
13
14          vector<pair<T, int>> assoc(n);
15          for (int i = 0; i < n; ++i) {
16              assoc[i] = make_pair(a[i], i);
17          }
18          sort(assoc.begin(), assoc.end());
19          for (int i = 0; i < n; ++i) {
20              sa[i] = assoc[i].second;
21              if (i == 0 || assoc[i].first != assoc[i - 1].first) {
22                  rank[sa[i]] = i;
23              } else {
24                  rank[sa[i]] = rank[sa[i - 1]];
25              }
26          }
27
28          vector<int> tmp(n), cnt(n);
29          vector<pair<int, int>> suffix(n);
30          for (int m = 1; m < n; m <<= 1) {
31              // snd
32              for (int i = 0; i < m; ++i) {
33                  tmp[i] = n - m + i;
34              }
35              for (int i = 0, j = m; i < n; ++i) {
36                  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;
            }
        }
    }
};
```

6.12 LU 分解

Listing 32: **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);
```

7 对一类问题的处理方法