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
inline void optimizeIO(void) {
    ios::sync_with_stdio(false);
    cin.tie(NULL), 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; }
void __print(long double x) { cerr << x; }
void __print(char x) { cerr << '\'' << x << '\''; }
void __print(const char *x) { cerr << '\"' << x << '\"'; }
void __print(const string &x) { cerr << '\"' << x << '\"'; }
void __print(bool x) { cerr << (x ? "true" : "false"); }
void __print(__int128_t x) {
    if (x < 0) cerr << '-', x = -x;
    if (x > 9) __print(x / 10);
    cerr << char((x % 10) ^ 48);
}
void dbgEndl(void) { cerr << '\n'; }

template <typename T, typename V>
void __print(const pair<T, V> &x) {
    cerr << '{', __print(x.first), cerr << ",␣", __print(x.second), cerr << '}';
}
template <typename T>
void __print(const T &x) {
    int f = 0;
    cerr << '{';
    for (auto i : x) cerr << (f++ ? ",␣" : ""), __print(i);
    cerr << "}";
}
void _print() { cerr << "]\n"; }
template <typename T, typename... V>
void _print(T t, V... v) {
    __print(t);
    if (sizeof...(v)) cerr << ",␣";
    _print(v...);
}
#ifdef DEBUG
// To customize a struct/class to print, just define the __print function.

#ifndef NO_DBG_COLOR
#define dbg(x...)                                                              \
    cerr << "\e[91m" << __func__ << ":" << __LINE__ << "␣[" << #x << "]␣=␣["; \
    _print(x);                                                                 \
    cerr << "\e[39m";

#define short_dbg(x...) \
    cerr << "\e[91m[["; \
    _print(x);          \
    cerr << "\e[39m";
#else
#define dbg(x...)                                                      \
    cerr << __func__ << ":" << __LINE__ << "␣[" << #x << "]␣=␣["; \
    _print(x);
#define short_dbg(x...) \
    cerr << "[";        \
    _print(x);
#endif // !NO_DBG_COLOR

#else
#define dbg(x...)
#endif
```

1.6 火车头

Listing 6: **optimize-header.h**

```
#pragma GCC optimize(3)
#pragma GCC target("avx")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
```

```
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

1.7 c-cpp-properties.json

Listing 7: **c-cpp-properties.json**

```
{
    "configurations": [
        {
            "name": "macos-gcc-arm64",
            "includePath": [
                "${workspaceFolder}/**",
                "/usr/local/include/ac-library/"
            ],
            "compilerPath": "/usr/local/bin/g++",
            "cStandard": "c17",
            "cppStandard": "c++20",
            "intelliSenseMode": "macos-gcc-arm64",
            "compilerArgs": [],
            "configurationProvider": "ms-vscode.makefile-tools"
        }
    ],
    "version": 4
}
```

1.8 launch.json

Listing 8: **launch.json**

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(lldb) Launch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}/compiled.out",
            "args": [],
            "stopAtEntry": false,
            "cwd": "dollar{fileDirname}",
            "environment": [],
            "externalConsole": true,
            "internalConsoleOptions": "neverOpen",
            "MIMode": "lldb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for lldb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": false
                }
            ],
            "preLaunchTask": "Compile"
        }
    ],
}
```

1.9 settings.json

Listing 9: **settings.json**

```
{
    "files.defaultLanguage": "cpp",
    "editor.formatOnType": true,
    "editor.suggest.snippetsPreventQuickSuggestions": false,
    "editor.acceptSuggestionOnEnter": "off",
    "C_Cpp.clang_format_sortIncludes": true,
    "C_Cpp.errorSquiggles": "disabled",
    "C_Cpp.default.defines": ["LOCAL", "DEBUG"]
}
```

1.10 tasks.json

Listing 10: **tasks.json**

```
// https://code.visualstudio.com/docs/editor/tasks
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "Compile", // 任务名称，与launch.json的preLaunchTask相对应
            "command": "g++", // 要使用的编译器，C++用g++
            "args": [
                "${file}",
                "-o", // 指定输出文件名，不加该参数则默认输出a.exe，Linux下默认a.out
                "${fileDirname}/compiled.out",
                "-g", // 生成和调试有关的信息
                // "-arch aarch64",
                // "-m64", // 不知为何有时会生成16位程序而无法运行，此条可强制生成64位的
                "-Wall", // 开启额外警告
                "-std=c++20", // c++14
                "-DLOCAL",
                "-DDEBUG",
                "-O3",
                "-ld_classic", // will be deprecated
                "-Wno-char-subscripts",
                "-I",
                "/usr/local/include/ac-library/"
                // "--stack=268435456"        // 手动扩大栈空间
            ], // 编译的命令，其实相当于VSC帮你在终端中输了这些东西
            "type": "process", // process是把预定义变量和转义解析后直接全部传给command；shell相当于先打开shell再输入命令，所以args还会经过shell再解析一遍
            "group": {
                "kind": "build",
                "isDefault": true // 不为true时ctrl shift B就要手动选择了
            },
            "presentation": {
                "echo": true,
                "reveal": "always", // 执行任务时是否跳转到终端面板，可以为always，silent，never。具体参见VSC的文档，即使设为never，手动点进去还是可以看到
                "focus": false, // 设为true后可以使执行task时焦点聚集在终端，但对编译C/C++来说，设为true没有意义
                "panel": "shared" // 不同的文件的编译信息共享一个终端面板
            },
            "problemMatcher": "$gcc" // 捕捉编译时终端里的报错信息到问题面板中，修改代码后需要重新编译才会再次触发
            // 本来有Lint，再开problemMatcher就有双重报错，但MinGW的Lint效果实在太差了；用Clangd可以注释掉
        }
    ]
}
```

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 11: **chtholly.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;
    }
};
```

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 12: **fenwick.cpp**

```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 13: **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 14: **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 15: **discretization.cpp**

```cpp
#include <vector>

template <class _Tp>
struct Mess {
    std::vector<_Tp> v;
    bool initialized = false;
    inline _Tp origin(int idx) { return v[idx - 1]; }
    inline void insert(_Tp 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(_Tp x) {
        if (!initialized) init();
        return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
    }
    inline bool exist(_Tp x) { return origin(query(x)) == x; }
};
```

2.6 并查集

Listing 16: **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;
    }
};
```

2.7 出现次数统计

O(nlogn) 预处理，O(logn) 查找的出现次数在线统计

Listing 17: **appear-statistics.cpp**

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

template <class _Tp>
struct Mess {
    std::vector<_Tp> v;
    bool initialized = false;
    inline _Tp origin(int idx) { return v[idx - 1]; }
    inline void insert(_Tp 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(_Tp x) {
        if (!initialized) init();
        return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
    }
    inline bool exist(_Tp x) { return origin(query(x)) == x; }
};

template <class T>
class AppearStats { // Appear Statistics.
private:
    Mess<T> M;
    size_t n;
    std::vector<std::vector<int>> pos;

public:
    AppearStats(void) : n(0) {}
    AppearStats(std::vector<T> &init) : n(init.size()) { _init(init); }
    inline void _init(std::vector<T> &init) {
        for (auto item : init) M.insert(item);
        n = init.size(), M.init(), pos.resize(M.v.size());
        for (size_t i = 0; i < n; i++) {
            pos[M.query(init[i]) - 1].push_back(i);
        }
    }
    // Use「base」as the beginning of index, return -1 if x doesn't exist.
    inline int first(int l, int r, T x, int base = 0) {
        l -= base, r -= base;
        if (!M.exist(x)) return -1;
        std::vector<int> &P = pos[M.query(x) - 1];
        auto it = std::lower_bound(P.begin(), P.end(), l);
        return it == P.end() || *it > r ? -1 : *it + base;
    }
    // Use「base」as the beginning of index, return -1 if x doesn't exist.
    inline int last(int l, int r, T x, int base = 0) {
        l -= base, r -= base;
        if (!M.exist(x)) return -1;
        std::vector<int> &P = pos[M.query(x) - 1];
        auto it = std::upper_bound(P.begin(), P.end(), r);
        return it == P.begin() || *std::prev(it) < l ? -1 : *std::prev(it) + base;
    }
    inline int count(int l, int r, T x, int base = 0) {
        l -= base, r -= base;
        if (!M.exist(x)) return 0;
        std::vector<int> &P = pos[M.query(x) - 1];
        auto L = std::lower_bound(P.begin(), P.end(), l);
        auto R = std::upper_bound(P.begin(), P.end(), r);
        if (L == P.end() || R == P.begin()) return 0;
        if (*L > r || *std::prev(R) < l) return 0;
        return R - L;
    }
};
```

3 数学（数论）算法

3.1 带模整数类

Listing 18: **mod-int.cpp**

```cpp
template <int mod>
inline int64_t down(int64_t x) { return x >= mod ? x - mod : x; }
template <int mod>
struct ModInt {
    int64_t x;
    ModInt() = default;
    ModInt(int64_t x) : x((x % mod + mod) % mod) {}
    friend istream &operator>>(istream &in, ModInt &a) { return in >> a.x; }
    friend ostream &operator<<(ostream &out, ModInt a) { return out << a.x; }
    friend ModInt operator+(ModInt a, ModInt b) { return down<mod>(a.x + b.x); }
    friend ModInt operator-(ModInt a, ModInt b) { return down<mod>(a.x - b.x + mod); }
    friend ModInt operator*(ModInt a, ModInt b) { return (__int128)a.x * b.x % mod; }
    friend ModInt operator/(ModInt a, ModInt b) { return a * ~b; }
    friend ModInt operator^(ModInt a, long long b) {
        ModInt ans = 1;
        for (; b; b >>= 1, a *= a)
            if (b & 1) ans *= a;
        return ans;
    }
    friend ModInt operator~(ModInt a) { return a ^ (mod - 2); }
    friend ModInt operator-(ModInt a) { return down<mod>(mod - a.x); }
    friend ModInt &operator+=(ModInt &a, ModInt b) { return a = a + b; }
    friend ModInt &operator-=(ModInt &a, ModInt b) { return a = a - b; }
    friend ModInt &operator*=(ModInt &a, ModInt b) { return a = a * b; }
    friend ModInt &operator/=(ModInt &a, ModInt b) { return a = a / b; }
    friend ModInt &operator^=(ModInt &a, long long b) { return a = a ^ b; }
    friend ModInt &operator++(ModInt &a) { return a += 1; }
    friend ModInt operator++(ModInt &a, int) {
        ModInt x = a;
        a += 1;
        return x;
    }
    friend ModInt &operator--(ModInt &a) { return a -= 1; }
    friend ModInt operator--(ModInt &a, int) {
        ModInt x = a;
        a -= 1;
        return x;
    }
    friend bool operator==(ModInt a, ModInt b) { return a.x == b.x; }
    friend bool operator!=(ModInt a, ModInt b) { return !(a == b); }
};
using mint = ModInt<>;
```

3.2 计算几何

Listing 19: **computation-geometry.cpp**

```
template <class T>
struct Point {
    T x, y;
    Point(void) = default;
    Point(T X, T Y) : x(X), y(Y) {}
    inline bool operator==(const Point B) {
        return x == B.x && y == B.y;
    }
    friend std::ostream &operator<<(std::ostream &out, Point P) {
        return out << "(" << P.x << ",␣" << P.y << ")";
    }
    friend std::istream &operator>>(std::istream &in, Point &P) {
        return in >> P.x >> P.y;
    }
};
template <class T>
struct Line {
    T A, B, C; // Ax + By + C = 0
    Line(void) = default;
    Line(T a, T b, T c) : A(a), B(b), C(c) {} // Ax + By + C = 0
    Line(T k, T b) : A(k), B(-1), C(b) {}     // y = kx + b
};
template <class T>
inline int sign(T x) {
    return x == 0 ? 0 : (x < 0 ? -1 : 1);
}
template <class T>
inline bool parallel(Line<T> P, Line<T> Q) {
    return P.A * Q.B == P.B * Q.A;
}
template <class T>
inline Point<T> intersect(Line<T> P, Line<T> Q) {
    assert(!parallel(P, Q));
    return Point<T>{
        (P.C * Q.B - Q.C * P.B) / (Q.A * P.B - P.A * Q.B),
        (P.C * Q.A - Q.C * P.A) / (P.A * Q.B - Q.A * P.B)
    };
}
```

```cpp
template <class T>
inline Line<T> getLine(Point<T> P, Point<T> Q) {
    assert(!(P == Q));
    if (P.x == Q.x) {
        return Line<T>(-1, 0, P.x);
    } else if (P.y == Q.y) {
        return Line<T>(0, -1, P.y);
    } else {
        const T k = (P.x - Q.x) / (P.y - Q.y);
        return Line<T>(k, P.y - k * P.x);
    }
}
template <class T>
inline bool pointOnLine(Point<T> P, Line<T> L) {
    return L.A * P.x + L.B + P.y + L.C == 0;
}
```

4 字符串算法

4.1 字符串哈希

Listing 20: **hashed-string.cpp**

```cpp
using i64 = long long;
using i128 = __int128;
class HashedString {
private:
    // change M and B if you want
    static const i64 M = (1LL << 61) - 1;
    static const i64 B;
    // pow[i] contains B^i % M
    static std::vector<i64> pow;
    // p_hash[i] is the hash of the first i characters of the given string
    std::vector<i64> r_hash, p_hash;
    i128 mul(i64 a, i64 b) { return (i128)a * b; }
    i64 mod_mul(i64 a, i64 b) { return mul(a, b) % M; }

public:
    HashedString(const string &s) : r_hash(s.size() + 1), p_hash(s.size() + 1) {
        while (pow.size() < s.size()) { pow.push_back(mod_mul(pow.back(), B)); }
        p_hash[0] = 0;
        r_hash[0] = 0;
        for (size_t i = 0; i < s.size(); i++) {
            p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M; // 1-based
        }
        i64 sz = s.size();
        for (int i = sz - 1, j = 0; i >= 0; i--, j++) {
            r_hash[j + 1] = (mul(r_hash[j], B) + s[i]) % M;
        }
    }
    i64 getHash(int start, int end) { // 0 based
        i64 raw_val = p_hash[end + 1] - mod_mul(p_hash[start], pow[end - start + 1]);
        return (raw_val + M) % M;
    }
    i64 getRHash(int start, int end) { // 0 based
        i64 raw_val = r_hash[end + 1] - mod_mul(r_hash[start], pow[end - start + 1]);
        return (raw_val + M) % M;
    }
};
std::vector<i64> HashedString::pow = {1};
mt19937 rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
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 21: **others/i128-stream.cpp**

```cpp
#include <iostream>

using i128 = __int128;

std::istream &operator>>(std::istream is, i128 &n) {
    std::string s;
    is >> s;
    for (auto c : s) {
        n = n * 10 + (c - '0');
    }
    return is;
}

std::ostream &operator<<(std::ostream &os, i128 n) {
    std::string s;
    while (n) {
        s += '0' + n % 10;
        n /= 10;
    }
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

5.2 常用数学运算库函数及 gcd 重载

Listing 22: **others/clf.cpp**

```cpp
using i64 = long long;
using i128 = __int128;
inline i64 ceilDiv(i64 n, i64 m) {
    if (n >= 0) {
        return (n + m - 1) / m;
    } else {
        return n / m;
    }
}
inline i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
    } else {
        return (n - m + 1) / m;
    }
}
template <class T>
inline void chmax(T &a, T b) {
    if (a < b) a = b;
}
template <class T>
inline void chmin(T &a, T b) {
    if (!(a < b)) a = b;
}
inline i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}
```

5.3 强连通分量缩点 (SCC)

Listing 23: **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 24: **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;
    }
```

```cpp

    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 25: **graph/bigraph-weight-match.cpp**

```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);
        }

        auto update = [&](int x) {
            for (int y = 0; y < ny; ++y) {
                if (lx[x] + ly[y] - a[x][y] < slack[y]) {
                    slack[y] = lx[x] + ly[y] - a[x][y];
                    slackx[y] = x;
                }
            }
        };

        costs.resize(nx + 1);
        costs[0] = 0;
        lx.assign(nx, std::numeric_limits<T>::max());
        ly.assign(ny, 0);
        xy.assign(nx, -1);
        yx.assign(ny, -1);
        slackx.resize(ny);
        for (int cur = 0; cur < nx; ++cur) {
            std::queue<int> que;
            visx.assign(nx, false);
            visy.assign(ny, false);
            slack.assign(ny, std::numeric_limits<T>::max());
            p.assign(nx, -1);

            for (int x = 0; x < nx; ++x) {
                if (xy[x] == -1) {
                    que.push(x);
                    visx[x] = true;
                    update(x);
                }
            }

            int ex, ey;
            bool found = false;
            while (!found) {
                while (!que.empty() && !found) {
                    auto x = que.front();
                    que.pop();
                    for (int y = 0; y < ny; ++y) {
                        if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
                            if (yx[y] == -1) {
                                ex = x;
                                ey = y;
                                found = true;
                                break;
                            }
                            que.push(yx[y]);
                            p[yx[y]] = x;
                            visy[y] = visx[yx[y]] = true;
                            update(yx[y]);
                        }
                    }
                }
                if (found)
                    break;

                T delta = std::numeric_limits<T>::max();
                for (int y = 0; y < ny; ++y)
                    if (!visy[y])
                        delta = std::min(delta, slack[y]);
                for (int x = 0; x < nx; ++x)
                    if (visx[x])
                        lx[x] -= delta;
                for (int y = 0; y < ny; ++y) {
                    if (visy[y]) {
                        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 26: **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);

        // disjoint set union
        auto find = [&](int u) {
            while (f[u] != u)
                u = f[u] = f[f[u]];
            return u;
        };

        auto lca = [&](int u, int v) {
            u = find(u);
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v])
                    std::swap(u, v);
                u = find(link[match[u]]);
            }
            return u;
        };

        std::queue<int> que;
        auto blossom = [&](int u, int v, int p) {
            while (find(u) != p) {
                link[u] = v;
```

```cpp
                v = match[u];
                if (vis[v] == 0) {
                    vis[v] = 1;
                    que.push(v);
                }
                f[u] = f[v] = p;
                u = link[v];
            }
        };

        // find an augmenting path starting from u and augment (if exist)
        auto augment = [&](int u) {
            while (!que.empty())
                que.pop();

            std::iota(f.begin(), f.end(), 0);

            // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
            std::fill(vis.begin(), vis.end(), -1);

            que.push(u);
            vis[u] = 1;
            dep[u] = 0;

            while (!que.empty()) {
                int u = que.front();
                que.pop();
                for (auto v : e[u]) {
                    if (vis[v] == -1) {

                        vis[v] = 0;
                        link[v] = u;
                        dep[v] = dep[u] + 1;

                        // found an augmenting path
                        if (match[v] == -1) {
                            for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
                                temp = match[y];
                                match[x] = y;
                                match[y] = x;
                            }
                            return;
                        }

                        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 27: **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; }
};
```

5.8 最大流

Listing 28: **graph/max-flow.cpp**

```
constexpr int inf = 1E9;
template <class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;

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

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
    }

    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }

    std::vector<bool> minCut() {
        std::vector<bool> c(n);
        for (int i = 0; i < n; i++) {
            c[i] = (h[i] != -1);
        }
        return c;
    }

    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

5.9 最小费用可行流（或最大流）

Listing 29: **graph/max-cost-flow-graph.cpp**

```
struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<i64> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<i64>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>, std::greater<std::pair<i64, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            i64 d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] < d) continue;
            for (int i : g[u]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<i64>::max();
    }
    MCFGraph(int n) : n(n), g(n) {}
    void addEdge(int u, int v, int c, int f) { // 可行流
        if (f < 0) {
            g[u].push_back(e.size());
            e.emplace_back(v, 0, f);
            g[v].push_back(e.size());
            e.emplace_back(u, c, -f);
        } else {
            g[u].push_back(e.size());
            e.emplace_back(v, c, f);
            g[v].push_back(e.size());
            e.emplace_back(u, 0, -f);
        }
    }
    // void addEdge(int u, int v, int c, int f) { // 最大流
    //     g[u].push_back(e.size());
    //     e.emplace_back(v, c, f);
    //     g[v].push_back(e.size());
    //     e.emplace_back(u, 0, -f);
    // }
    std::pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            int aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug, e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
        }
        return std::make_pair(flow, cost);
    }
};
```

5.10 树链剖分

Listing 30: **graph/hld.cpp**

```
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    bool isAncestor(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }

    int rootedParent(int u, int v) {
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncestor(u, v)) {
            return parent[u];
        }
        auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncestor(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

5.11 快速幂

Listing 31: **math/fast-pow.cpp**

```
int power(int a, i64 b, int p) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % p) {
        if (b % 2) {
            res = 1LL * res * a % p;
        }
    }
    return res;
}
```

5.12 欧拉筛

Listing 32: **math/euler-sieve.cpp**

```
std::vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

5.13 单点欧拉函数

Listing 33: **math/phi.cpp**

```
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res = res / i * (i - 1);
        }
    }
    if (n > 1) {
        res = res / n * (n - 1);
    }
    return res;
}
```

5.14 exgcd

Listing 34: **math/exgcd.cpp**

```
int exgcd(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
```

5.15 组合数

Listing 35: **math/comb.cpp**

```cpp
struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        m = std::min(m, Z::getMod() - 1);
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;
```

5.16 树状数组

Listing 36: **ds/fenwick.cpp**

```cpp
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int select(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i - 1] <= k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

5.17 Splay

Listing 37: **ds/splay.cpp**

```
struct Node {
    Node *l = nullptr;
    Node *r = nullptr;
    int cnt = 0;
    i64 sum = 0;
};

Node *add(Node *t, int l, int r, int p, int v) {
    Node *x = new Node;
    if (t) {
        *x = *t;
    }
    x->cnt += 1;
    x->sum += v;
    if (r - l == 1) {
        return x;
    }
    int m = (l + r) / 2;
    if (p < m) {
        x->l = add(x->l, l, m, p, v);
    } else {
        x->r = add(x->r, m, r, p, v);
    }
    return x;
}

int find(Node *tl, Node *tr, int l, int r, int x) {
    if (r <= x) {
        return -1;
    }
    if (l >= x) {
        int cnt = (tr ? tr->cnt : 0) - (tl ? tl->cnt : 0);
        if (cnt == 0) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
    }
    int m = (l + r) / 2;
    int res = find(tl ? tl->l : tl, tr ? tr->l : tr, l, m, x);
    if (res == -1) {
        res = find(tl ? tl->r : tl, tr ? tr->r : tr, m, r, x);
    }
    return res;
}

std::pair<int, i64> get(Node *t, int l, int r, int x, int y) {
    if (l >= y || r <= x || !t) {
        return {0, 0LL};
    }
    if (l >= x && r <= y) {
        return {t->cnt, t->sum};
    }
    int m = (l + r) / 2;
    auto [cl, sl] = get(t->l, l, m, x, y);
    auto [cr, sr] = get(t->r, m, r, x, y);
    return {cl + cr, sl + sr};
}

struct Tree {
    int add = 0;
    int val = 0;
    int id = 0;
    Tree *ch[2] = {};
    Tree *p = nullptr;
};

int pos(Tree *t) {
    return t->p->ch[1] == t;
}

void add(Tree *t, int v) {
    t->val += v;
    t->add += v;
}

void push(Tree *t) {
    if (t->ch[0]) {
        add(t->ch[0], t->add);
    }
    if (t->ch[1]) {
        add(t->ch[1], t->add);
    }
    t->add = 0;
}

void rotate(Tree *t) {
    Tree *q = t->p;
    int x = !pos(t);
    q->ch[!x] = t->ch[x];
    if (t->ch[x]) t->ch[x]->p = q;
    t->p = q->p;
    if (q->p) q->p->ch[pos(q)] = t;
    t->ch[x] = q;
    q->p = t;
}

void splay(Tree *t) {
    std::vector<Tree *> s;
    for (Tree *i = t; i->p; i = i->p) s.push_back(i->p);
    while (!s.empty()) {
        push(s.back());
        s.pop_back();
    }
    push(t);
    while (t->p) {
        if (t->p->p) {
            if (pos(t) == pos(t->p)) rotate(t->p);
            else rotate(t);
        }
        rotate(t);
    }
}

void insert(Tree *&t, Tree *x, Tree *p = nullptr) {
    if (!t) {
        t = x;
        x->p = p;
        return;
    }

    push(t);
    if (x->val < t->val) {
        insert(t->ch[0], x, t);
    } else {
        insert(t->ch[1], x, t);
    }
}

void dfs(Tree *t) {
    if (!t) {
        return;
    }
    push(t);
    dfs(t->ch[0]);
    std::cerr << t->val << "␣";
    dfs(t->ch[1]);
}

std::pair<Tree *, Tree *> split(Tree *t, int x) {
    if (!t) {
        return {t, t};
    }
    Tree *v = nullptr;
    Tree *j = t;
    for (Tree *i = t; i;) {
        push(i);
        j = i;
        if (i->val >= x) {
            v = i;
            i = i->ch[0];
        } else {
            i = i->ch[1];
        }
    }

    splay(j);
    if (!v) {
        return {j, nullptr};
    }

    splay(v);

    Tree *u = v->ch[0];
    if (u) {
        v->ch[0] = u->p = nullptr;
    }
    // std::cerr << "split " << x << "\n";
    // dfs(u);
    // std::cerr << "\n";
    // dfs(v);
    // std::cerr << "\n";
    return {u, v};
}

Tree *merge(Tree *l, Tree *r) {
    if (!l) {
        return r;
    }
    if (!r) {
        return l;
    }
    Tree *i = l;
    while (i->ch[1]) {
        i = i->ch[1];
    }
    splay(i);
    i->ch[1] = r;
    r->p = i;
    return i;
}
```

5.18 取模类，按需写

Listing 38: **ds/mod.cpp**

```
template <class T>
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr i64 mul(i64 a, i64 b, i64 p) {
    i64 res = a * b - i64(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template <i64 P>
struct MLong {
    i64 x;
    constexpr MLong() : x{} {}
    constexpr MLong(i64 x) : x{norm(x % getMod())} {}

    static i64 Mod;
    constexpr static i64 getMod() {
        if (P > 0) {
            return P;
```

```cpp
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(i64 Mod_) {
        Mod = Mod_;
    }
    constexpr i64 norm(i64 x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr i64 val() const {
        return x;
    }
    explicit constexpr operator i64() const {
        return x;
    }
    constexpr MLong operator-() const {
        MLong res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MLong inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MLong &operator*=(MLong rhs) & {
        x = mul(x, rhs.x, getMod());
        return *this;
    }
    constexpr MLong &operator+=(MLong rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MLong &operator-=(MLong rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MLong &operator/=(MLong rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MLong operator*(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MLong operator+(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MLong operator-(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MLong operator/(MLong lhs, MLong rhs) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
        i64 v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MLong lhs, MLong rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
        return lhs.val() != rhs.val();
    }
};

template <>
i64 MLong<0LL>::Mod = i64(1E18) + 9;

template <int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template <>
int MInt<0>::Mod = 998244353;

template <int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 1000000007;
using Z = MInt<P>;
```

5.19 马拉车

Listing 39: **string/manacher.cpp**

```
std::vector<int> manacher(std::vector<int> s) {
    std::vector<int> t{0};
    for (auto c : s) {
        t.push_back(c);
        t.push_back(0);
    }
    int n = t.size();
    std::vector<int> r(n);
    for (int i = 0, j = 0; i < n; i++) {
        if (2 * j - i >= 0 && j + r[j] > i) {
            r[i] = std::min(r[2 * j - i], j + r[j] - i);
        }
        while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
            r[i] += 1;
        }
        if (i + r[i] > j + r[j]) {
            j = i;
        }
    }
    return r;
}
```

5.20 Z 函数

Listing 40: **string/z-func.cpp**

```
std::vector<int> zFunction(std::string s) {
    int n = s.size();
    std::vector<int> z(n + 1);
    z[0] = n;
    for (int i = 1, j = 1; i < n; i++) {
        z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > j + z[j]) {
            j = i;
        }
    }
    return z;
}
```

5.21 SA 后缀数组

Listing 41: **string/suffix-array.cpp**

```
struct SuffixArray {
    int n;
    std::vector<int> sa, rk, lc;
    SuffixArray(const std::string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(), [&](int a, int b) { return s[a] < s[b]; });
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; ++i)
                tmp.push_back(n - k + i);
            for (auto i : sa)
                if (i >= k)
                    tmp.push_back(i - k);
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; ++i)
```

```
                ++cnt[rk[i]];
            for (int i = 1; i < n; ++i)
                cnt[i] += cnt[i - 1];
            for (int i = n - 1; i >= 0; --i)
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            std::swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; ++i)
                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; ++i) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j];)
                    ++j;
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

6 Watashi 代码库 (备用)

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

Listing 42: **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 43: **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 44: **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 45: **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;
    }
};
```

6.5 轻重权树剖分

Listing 46: **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;
```

6.6 强连通分量

Listing 47: **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]);
            } 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();
    }
};
```

6.7 双连通分量

Listing 48: **bcc.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;
```

```cpp

    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);
    }
};
```

6.8 二分图匹配

Listing 49: **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;
    }
};
```

6.9 最小费用最大流

Listing 50: **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);
    }
};
```

6.10 AhoCorasick 自动机

Listing 51: **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;
```

6.11 后缀数组

Listing 52: **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;
            }
        }
    }
};
```

6.12 LU 分解

Listing 53: **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