Alfred
代码模版库

目录

1 比赛配置 and 奇技淫巧 **4**
1.1 多组数据代码模板 4
1.2 快读快写 4
1.3 关闭流与 C 风格输入输出的同步 4
1.4 .clang-format 5
1.5 debug.h 5
1.6 火车头 7
1.7 c-cpp-properties.json 7
1.8 launch.json 8
1.9 settings.json 8
1.10 tasks.json 8

2 数据结构 **9**
2.1 珂朵莉树 9
2.2 树状数组 10
2.3 静态可重区间信息（支持 RMQ） 11
2.4 可删除优先队列 12
2.5 PBDS 大常数平衡树 12
2.6 离散化容器 13
2.7 并查集 13
2.8 可撤销并查集 14
2.9 带权并查集 14
2.10 出现次数统计 15
2.11 01-Trie 16
2.12 滑动窗口 17
2.13 （二维）前缀和 18

3 数学（数论）算法 **19**
3.1 带模整数类 19
3.2 计算几何 20
3.3 组合数学 21
3.4 拉格朗日插值 22
3.5 光速幂 23

4 字符串算法 **23**
4.1 字符串哈希 23

5 图论 **24**
5.1 模块化基本图 24
5.2 O(1) LCA 25

6 jiangly 代码库 (备用，侵权请提出 issue) **25**
6.1 01 - int128 库函数自定义 25
6.2 02 - 常用库函数重载 26
6.3 03 - 字符调整 27
6.4 04A - 二分算法（整数域） 28
6.5 04B - 二分算法（实数域） 28
6.6 01 - 强连通分量缩点（SCC） 29
6.7 02 - 割边与割边缩点（EBCC） 30
6.8 03 - 二分图最大权匹配（MaxAssignment 基于 KM） 32
6.9 04 - 一般图最大匹配（Graph 带花树算法）【久远】 34
6.10 05 - TwoSat（2-Sat） 36
6.11 06A - 最大流（Flow 旧版其一，整数应用） 37
6.12 06B - 最大流（Flow 旧版其二，浮点数应用） 38
6.13 06C - 最大流（MaxFlow 新版） 39
6.14 07A - 费用流（MCFGraph 旧版） . . . 41
6.15 07B - 费用流（MinCostFlow 新版） . . 42
6.16 08 - 树链剖分（HLD） 44
6.17 01 - 快速幂 46
6.18 02 - 基姆拉尔森公式 46
6.19 03 - 欧拉筛 47
6.20 04 - 莫比乌斯函数筛（莫比乌斯反演） . 47
6.21 05 - 扩展欧几里得（exgcd） 48
6.22 06A - 欧拉函数（求解单个数的欧拉函数） 49
6.23 06B - 欧拉函数（求解全部数的欧拉函数） 49
6.24 07A - 组合数（小范围预处理，逆元 + 杨辉三角） 50
6.25 07B - 组合数（Comb, with. ModIntBase） 50
6.26 08 - 素数测试与因式分解（Miller-Rabin and Pollard-Rho） 51
6.27 09A - 平面几何（Point） 53
6.28 09B - 平面几何（with. std::complex） . 59
6.29 10 - 立体几何（Point） 59
6.30 11A - 静态凸包（with. Point，旧版） . 60
6.31 11B - 静态凸包（with. Point，新版） . 62
6.32 11C - 静态凸包（with. std::complex） . 63
6.33 12A - 多项式（Poly, with. Z） 64
6.34 12B - 多项式（Poly, with. MInt） . . . 68
6.35 12C - 多项式乘法 75
6.36 13A - 生成函数（q-int） 77
6.37 13B - 生成函数（q-Binomial） 77
6.38 13C - 生成函数（Binomial 任意模数二项式） 78
6.39 14 - 自适应辛普森法 Simpson 80
6.40 15 - 矩阵（Matrix） 81
6.41 16 - 高斯消元（guess）【久远】 82
6.42 01A - 树状数组（Fenwick 旧版） 82
6.43 01B - 树状数组（Fenwick 新版） 83
6.44 02 - 并查集（DSU） 84
6.45 03A - 线段树（SegmentTree+Info 区间加 + 单点修改） 85
6.46 03B - 线段树（SegmentTree 区间乘 + 单点加） 86
6.47 03C - 线段树（SegmentTree+Info 初始赋值 + 单点修改 + 查找前驱后继） . . 87
6.48 03D - 线段树（SegmentTree+Info+Merge 初始赋值 + 单点修改 + 区间合并） . . 89
6.49 04 - 懒标记线段树（LazySegmentTree） 90
6.50 05A - 取模类（Z 旧版） 93
6.51 05B - 取模类（MLong and MInt 新版） 95
6.52 05C - 动态取模类（ModIntBase） . . . 98
6.53 06 - 状压 RMQ（RMQ） 100
6.54 07A - Splay 102
6.55 07B - Splay 105
6.56 07C - Splay 106
6.57 08A - 其他平衡树 110
6.58 08B - 其他平衡树 111
6.59 08C - 其他平衡树 112
6.60 08D - 其他平衡树 114
6.61 09 - 分数四则运算（Frac） 116
6.62 10 - 线性基（Basis） 117
6.63 143 - 高精度（BigInt） 117

7 Watashi 代码库 (备用) 118
7.1 $O(n\log n)-O(1)$ RMQ 118
7.2 $O(n\log n)-O(\log n)$ LCA 119
7.3 树状数组 120
7.4 并查集 121
7.5 轻重权树剖分 121
7.6 强连通分量 123
7.7 双连通分量 124
7.8 二分图匹配 126
7.9 最小费用最大流 127
7.10 AhoCorasick 自动机 129
7.11 后缀数组 130
7.12 LU 分解 131

8 对一类问题的处理方法 132

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
#include <bits/stdc++.h>

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 <bits/stdc++.h>

```

```
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
InsertNewlineAtEOF: true
```

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 {
```

```
26          cerr << "inf";
27      }
28  }
29  void __print(unsigned x) { cerr << x; }
30  void __print(unsigned long x) { cerr << x; }
31  void __print(unsigned long long x) { cerr << x; }
32  void __print(float x) { cerr << x; }
33  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
```

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**

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

struct ChthollyTree {
    using i64 = long long;
    struct Node {
        mutable i64 l, r, v;
        inline bool operator<(const Node &x) const { return l < x.l; }
    };
    std::set<Node> tr;
    using iterator = std::set<Node>::iterator;
    ChthollyTree(void) = default;
    ChthollyTree(int rng, int val) { init(rng, val); }
    inline void init(i64 rng, i64 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(i64 pos) {
        auto it = tr.lower_bound({pos, 0, 0});
        if (it != tr.end() && it->l == pos) return it;
        i64 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(i64 l, i64 r, i64 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(i64 l, i64 r, _Functor func) {
        auto R = split(r + 1), L = split(l);
        for (auto it = L; it != R; it++) func(it);
    }
    template <class _Functor> // func(i64 &, iterator)
    inline i64 query(i64 l, i64 r, _Functor func) {
        i64 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**

```
#include "fenwick.h"
#include <bits/stdc++.h>

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 l == 0 ? query(r) : subtract(query(r), query(l - 1));
    }
    inline int kth(const T k) {
        int ans = 0;
        for (int i = 1 << std::__lg(c.size() - 1); i; i >>= 1) {
            if (ans + i < (int)c.size() && c[ans + i] <= k) {
                k -= c[ans + i], ans += i;
```

```
            }
        }
        return ans;
    }
    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**

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

template <class T>
struct MaxInfo {
    T val;
    MaxInfo(void) { val = std::numeric_limits<T>::min(); }
    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(void) { val = std::numeric_limits<T>::max(); }
    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(void) { val = T(); }
    template <class InitT>
    GcdInfo(InitT x) { val = x; }
    GcdInfo operator+(GcdInfo &x) {
#if __cplusplus >= 201703L
        return {std::gcd(x.val, val)};
#else
        return {__gcd(x.val, val)};
#endif
    }
};
template <class T>
class SparseTable {
private:
    int n;
    std::vector<std::vector<T>> ST;

public:
    SparseTable(void) {}
    SparseTable(int N) : n(N), ST(N, std::vector<T>(std::__lg(N) + 1)) {}
    template <class InitT>
    SparseTable(std::vector<InitT> &_init) : SparseTable(_init.size()) { init(_init, true); }
    template <class InitT>
    inline void init(std::vector<InitT> &_init, bool internal = false) {
        if (!internal) {
            n = _init.size();
```

```
            ST.assign(n, std::vector<T>(std::__lg(n) + 1));
        }
        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
        if (l > r) return T();
        int w = std::__lg(r - l + 1);
        return ST[l][w] + ST[r - (1 << w) + 1][w];
    }
    inline T disjoint_query(int l, int r) {
        T ans = T();
        for (int i = l; i <= r; i += (1 << std::__lg(r - i + 1))) {
            ans = ans + ST[i][std::__lg(r - i + 1)];
        }
        return ans;
    }
};
```

2.4 可删除优先队列

基于优先队列，支持查询最大值，在线插入删除的数据结构，常数优于 std::set。

Listing 14: **priority-set.cpp**

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

template <class T, class Comp = std::less<T>>
class PrioritySet { // warning: all erase operations must be legal.
private:
    std::priority_queue<T, std::vector<T>, Comp> data;
    std::priority_queue<T, std::vector<T>, Comp> erased;

public:
    explicit PrioritySet(void) : data(), erased() {};
    explicit PrioritySet(std::vector<T> &init) {
        for (auto &v : init) {
            insert(v);
        }
    }
    inline void insert(const T &&x) { data.push(x); }
    inline void erase(const T &&x) { erased.push(x); }
    inline T &top(void) noexcept {
        assert(data.size() >= erased.size());
        while (!erased.empty() && data.top() == erased.top()) {
            data.pop(), erased.pop();
        }
        return data.top();
    }
    inline size_t size(void) {
        return data.size() - erased.size();
    }
};
```

2.5 PBDS 大常数平衡树

GNU PBDS 提供的大常数基于 rb-tree 的平衡树。

Listing 15: **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>;
```

2.6 离散化容器

Listing 16: **discretization.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 void clear(void) { v.clear(), initialized = false; }
    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; }
};
```

2.7 并查集

Listing 17: **dsu.cpp**

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

struct DSU {
    std::vector<int> fa, siz;
    DSU(int n) : fa(n + 1), siz(n + 1, 1) {
        std::iota(fa.begin(), fa.end(), 0);
    }
    inline int find(int x) {
        return fa[x] == x ? x : fa[x] = find(fa[x]);
    }
    inline bool same(int x, int y) {
        return find(x) == find(y);
    }
    // true if x and y were not in the same set, false otherwise.
    inline bool merge(int x, int y) {
        int fx = find(x), fy = find(y);
        if (fx == fy) return false;
        if (siz[fx] < siz[fy]) swap(fx, fy);
        fa[fy] = fx, siz[fx] += siz[fy], siz[fy] = 0;
        return true;
    }
```

```
    // x -> y, a.k.a let x be son of y (disable merge by rank).
    inline bool directed_merge(int x, int y) {
        int fx = find(x), fy = find(y);
        if (fx == fy) return false;
        fa[fx] = fy, siz[fy] += siz[fx], siz[fx] = 0;
        return true;
    }
};
```

2.8 可撤销并查集

Listing 18: **cancel-dsu.cpp**

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

struct CancelDSU {
    std::stack<int> S;
    std::vector<int> fa, siz;
    CancelDSU(int n) : fa(n + 1), siz(n + 1, 1) {
        std::iota(fa.begin(), fa.end(), 0);
    }
    inline int find(int x) {
        return fa[x] == x ? x : find(fa[x]);
    }
    inline bool same(int x, int y) {
        return find(x) == find(y);
    }
    inline void merge(int u, int v) {
        int fu = find(u), fv = find(v);
        if (fu == fv) return S.push(-1);
        if (siz[fu] < siz[fv]) swap(fu, fv);
        siz[fu] += siz[fv], fa[fv] = fu, S.push(fv);
    }
    inline void _cancel(void) {
        if (S.empty()) return;
        if (S.top() == -1) return S.pop();
        siz[fa[S.top()]] -= siz[S.top()];
        fa[S.top()] = S.top(), S.pop();
    }
    inline void cancel(int t = 1) {
        while (t--) _cancel();
    }
};
```

2.9 带权并查集

Listing 19: **weighted-dsu.cpp**

```
template <class T>
struct WeightedDSU {
    std::vector<int> fa;
    std::vector<T> w;
    WeightedDSU(int n) : fa(n + 1), w(n + 1) {
        std::iota(fa.begin(), fa.end(), 0);
    }
    inline int find(int x) {
        if (fa[x] == x) return x;
        int f = fa[x], f2 = find(f);
        return w[x] += w[f], fa[x] = f2;
    }
    inline bool same(int x, int y) {
        return find(x) == find(y);
```

```
    }
    // Given info: a[x] + v = a[y]
    // Returns true if this operation has no conflict, false otherwise.
    inline bool merge(int x, int y, T v) {
        int fx = find(x), fy = find(y);
        if (fx == fy) {
            return w[x] + v == w[y];
        }
        w[fy] = w[x] + v - w[y], fa[fy] = fx;
        return true;
    }
    inline T distance(int x, int y) {
        return find(x), find(y), w[y] - w[x];
    }
};
```

2.10 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 20: **appear-statistics.cpp**

```
#include "discretization.h"
#include <bits/stdc++.h>

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);
```

```
42          auto R = std::upper_bound(P.begin(), P.end(), r);
43          if (L == P.end() || R == P.begin()) return 0;
44          if (*L > r || *std::prev(R) < l) return 0;
45          return R - L;
46      }
47  };
```

2.11 01-Trie

Listing 21: **binary-trie.cpp**

```
1   // Thanks neal for this template.
2   #include <bits/stdc++.h>
3
4   const int BITS = 30;
5   const int INF = 1e9 + 7;
6   struct BinaryTrie { // 01-Trie
7       static const int ALPHABET = 2;
8       struct Node {
9           const int parent;
10          int words_here = 0;    // How many words EXACTLY here.
11          int starting_with = 0; // How many words have the PREFIX of this node.
12          int min_index = INF;   // The minimum index of words which have PREFIX of this node.
13          int max_index = -INF;  // The maximum index of words which have PREFIX of this node.
14          std::array<int, ALPHABET> child;
15          Node(int p = -1) : parent(p) { child.fill(-1); }
16      };
17      static const int ROOT = 0;
18      std::vector<Node> tr = {Node()};
19      BinaryTrie(int total_length = -1) { // Sum of |s|, leave -1 if don't know.
20          if (total_length >= 0) tr.reserve(total_length + 1);
21      }
22      // Returns the Node reference of word.
23      // NOTICE: this function creates a new Node if word isn't in the trie.
24      Node &operator[](uint64_t word) {
25          return tr[build(word, 0)];
26      }
27      // Get or create c-th (c = 0, 1) child of node
28      // Returns BinaryTrie node.
29      int get_or_create_child(int node, int c) {
30          if (tr[node].child[c] == -1) {
31              tr[node].child[c] = (int)tr.size();
32              tr.push_back(Node(node));
33          }
34          return tr[node].child[c];
35      }
36      // Build rootpath of word, insert delta (个) words
37      // Returns BinaryTrie node.
38      int build(uint64_t word, int delta) {
39          int node = ROOT;
40          for (int i = BITS - 1; i >= 0; i--) {
41              tr[node].starting_with += delta;
42              node = get_or_create_child(node, word >> i & 1);
43          }
44          tr[node].starting_with += delta;
45          return node;
46      }
47      // Insert a word with the index of index, INF if index is unknown.
48      // Returns BinaryTrie node.
49      int insert(uint64_t word, int index = INF) {
50          int node = build(word, 1);
51          tr[node].words_here += 1;
52          for (int x = node; x != -1; x = tr[x].parent) {
53              if (index != INF) {
```

```
                tr[x].min_index = std::min(tr[x].min_index, index);
                tr[x].max_index = std::max(tr[x].max_index, index);
            }
        }
        return node;
    }
    // Find such an x inserted in the trie that word ^ x is minimized.
    // Returns such x (x is certain).
    uint64_t query_min(uint64_t word) {
        int node = ROOT;
        uint64_t val = 0;
        for (int i = BITS - 1; i >= 0; i--) {
            int go_bit = word >> i & 1;
            if (tr[node].child[go_bit] == -1) {
                go_bit ^= 1;
            }
            val |= 1ull << go_bit;
            node = tr[node].child[go_bit];
        }
        return val;
    }
    // Find such an x inserted in the trie that word ^ x is maximized.
    // Returns such x (x is certain).
    uint64_t query_max(uint64_t word) {
        int node = ROOT;
        uint64_t val = 0;
        for (int i = BITS - 1; i >= 0; i--) {
            int go_bit = (word >> i & 1) ^ 1;
            if (tr[node].child[go_bit] == -1) {
                go_bit ^= 1;
            }
            val |= 1ull << go_bit;
            node = tr[node].child[go_bit];
        }
        return val;
    }
    // CF1983F: Find such an x inserted in the trie that word ^ x < upper_bound
    // Returns a pair {min_index, max_index} of x.
    std::pair<int, int> query_ub(uint64_t word, uint64_t upper_bound) {
        int mn = INF, mx = -INF, node = ROOT;
        for (int i = BITS - 1; i >= 0; i--) {
            int word_bit = word >> i & 1;      // digit i of word
            int ub_bit = upper_bound >> i & 1; // digit i of ub
            if (ub_bit == 1 && tr[node].child[word_bit] != -1) {
                // if digit i of ub is 1, then we can choose either
                // the subtree of word_bit or word_bit ^ 1.
                mn = std::min(mn, tr[tr[node].child[word_bit]].min_index);
                mx = std::max(mx, tr[tr[node].child[word_bit]].max_index);
            }
            // else if digit i of ub is 0, then we can only choose
            // the subtree of word_bit. (otherwise, we will violate the range)
            node = tr[node].child[word_bit ^ ub_bit];
            if (node == -1) break;
        }
        return {mn, mx};
    }
};
```

2.12 滑动窗口

Listing 22: **sliding-window.cpp**

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

```
2
3  template <class T> // default max.
4  std::vector<T> sliding_window(std::vector<T> A, size_t k) {
5      std::vector<T> res;
6      std::deque<size_t> Q;
7      for (size_t i = 0; i < A.size(); i++) {
8          if (!Q.empty() && Q[0] + k == i) {
9              Q.pop_front();
10         }
11         while (!Q.empty() && A[Q.back()] <= A[i]) {
12             Q.pop_back();
13         }
14         Q.push_back(i);
15         if (i >= k - 1) { // warning: assert k >= 1
16             res.push_back(A[Q[0]]);
17         }
18     }
19     return res;
20 }
21 template <class T>
22 std::vector<std::vector<T>> grid_sliding_window(
23     std::vector<std::vector<T>> &A, size_t x, size_t y
24 ) {
25     const size_t n = A.size(), m = A[0].size();
26     std::vector<std::vector<T>> cols(m - y + 1);
27     std::vector<std::vector<T>> ans(n - x + 1, std::vector<T>(m - y + 1));
28     for (size_t i = 0; i < n; i++) {
29         std::vector<T> res = sliding_window(A[i], y);
30         for (size_t j = 0; j <= m - y; j++) {
31             cols[j].push_back(res[j]);
32         }
33     }
34     for (size_t j = 0; j <= m - y; j++) {
35         std::vector<T> res = sliding_window(cols[j], x);
36         for (size_t i = 0; i <= n - x; i++) {
37             ans[i][j] = res[i];
38         }
39     }
40     return ans;
41 }
```

2.13 （二维）前缀和

Listing 23: **prefix-sum.cpp**

```
1  #include <bits/stdc++.h>
2
3  template <class T>
4  class Sum {
5  private:
6      size_t n;
7      std::vector<T> sum;
8
9  public:
10     Sum(void) : n(0) {}
11     template <class InitT>
12     Sum(std::vector<InitT> &init) { _init(init); }
13     template <class InitT>
14     inline void _init(std::vector<InitT> &init) {
15         if (init.empty()) return;
16         sum.resize(n = init.size()), sum[0] = init[0];
17         for (size_t i = 1; i < n; i++) {
18             sum[i] = sum[i - 1] + init[i];
```

```
        }
    }
    inline T query(int l, int r) {
        if (l > r) return T();
        return l == 0 ? sum[r] : sum[r] - sum[l - 1];
    }
};
template <class T>
class GridSum {
private:
    size_t n, m;
    std::vector<std::vector<T>> sum;

public:
    GridSum(void) : n(0), m(0) {}
    template <class InitT>
    GridSum(std::vector<std::vector<InitT>> &init) { _init(init); }
    template <class InitT>
    inline void _init(std::vector<std::vector<InitT>> &init) {
        if (init.empty()) return;
        n = init.size(), m = init[0].size();
        sum.assign(n, std::vector<T>(m)), sum[0][0] = init[0][0];
        for (size_t i = 1; i < n; i++) {
            sum[i][0] = sum[i - 1][0] + init[i][0];
        }
        for (size_t i = 1; i < m; i++) {
            sum[0][i] = sum[0][i - 1] + init[0][i];
        }
        for (size_t i = 1; i < n; i++) {
            for (size_t j = 1; j < m; j++) {
                sum[i][j] = sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1] + init[i][j];
            }
        }
    }
    inline T query(int x1, int y1, int x2, int y2) {
        T s1 = x1 == 0 ? 0 : sum[x1 - 1][y2];
        T s2 = y1 == 0 ? 0 : sum[x2][y1 - 1];
        T s3 = x1 == 0 || y1 == 0 ? 0 : sum[x1 - 1][y1 - 1];
        return sum[x2][y2] - s1 - s2 + s3;
    }
};
```

3 数学（数论）算法

3.1 带模整数类

Listing 24: **mod-int.cpp**

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

template <int mod>
inline int down(int x) { return x >= mod ? x - mod : x; }
template <int mod>
struct ModInt {
    int x;
    ModInt(void) = default;
    ModInt(int x) : x(x) {}
    ModInt(long long x) : x(x % mod) {}
    friend std::istream &operator>>(std::istream &in, ModInt &a) { return in >> a.x; }
    friend std::ostream &operator<<(std::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 (long long)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 > 0; 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); }
};
inline void __print(mint x) { std::cerr << x; }
```

3.2 计算几何

Listing 25: **computation-geometry.cpp**

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

// Caution: This computation geometry template is pure shit
//          because of the terrible math level of the author.
//          It will be rewritten some time.
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)
    };
}
template <class T>
inline Line<T> get_line(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 {
        return Line<T>(
            Q.y - P.y, P.x - Q.x, P.y * Q.x - P.x * Q.y
        );
    }
}
template <class T>
inline bool point_on_line(Point<T> P, Line<T> L) {
    return L.A * P.x + L.B * P.y + L.C == 0;
}
template <class T>
inline T dis_square(Point<T> P, Point<T> Q) {
    return (P.x - Q.x) * (P.x - Q.x) + (P.y - Q.y) * (P.y - Q.y);
}
```

3.3 组合数学

Listing 26: **comb.cpp**

```
#include "mod-int.h"
#include <bits/stdc++.h>

template <class mint>
struct Comb {
    int n;
    std::vector<mint> _fac, _invfac, _inv;
    Comb(void) : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() { init(n); }
    inline void init(int m) {
        _fac.resize(m + 1), _inv.resize(m + 1), _invfac.resize(m + 1);
        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = ~_fac[m];
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }
    inline mint fac(int m) {
```

```
23          if (m > n) init(m);
24          return _fac[m];
25      }
26      inline mint invfac(int m) {
27          if (m > n) init(m);
28          return _invfac[m];
29      }
30      inline mint inv(int m) {
31          if (m > n) init(m);
32          return _inv[m];
33      }
34      inline mint binom(int n, int m) {
35          if (n < m || m < 0) return 0;
36          return fac(n) * invfac(m) * invfac(n - m);
37      }
38  };
39  Comb<mint> comb;
```

3.4 拉格朗日插值

Listing 27: **lagrange.cpp**

```
1   #include "comb.h"
2   #include "mod-int.h"
3   #include <bits/stdc++.h>
4
5   inline mint lagrange(std::vector<mint> &x, std::vector<mint> &y, mint k) {
6       mint ans = 0, cur;
7       const int n = x.size();
8       for (int i = 0; i < n; i++) {
9           cur = y[i];
10          for (int j = 0; j < n; j++) {
11              if (j == i) continue;
12              cur *= (k - x[j]) / (x[i] - x[j]);
13          }
14          ans += cur;
15      }
16      return ans;
17  }
18  // y[0] is placeholder.
19  // If for all integer x_i in [1, n], we have f(x_i) = y_i (mod p), find f(k) mod p.
20  inline mint cont_lagrange(std::vector<mint> &y, mint k) {
21      mint ans = 0;
22      const int n = y.size() - 1;
23      std::vector<mint> pre(n + 1, 1), suf(n + 2, 1);
24      for (int i = 1; i <= n; i++) pre[i] = pre[i - 1] * (k - i);
25      for (int i = n; i >= 1; i--) suf[i] = suf[i + 1] * (k - i);
26      for (int i = 1; i <= n; i++) {
27          mint A = pre[i - 1] * suf[i + 1];
28          mint B = comb.fac(i - 1) * comb.fac(n - i);
29          ans += ((n - i) & 1 ? -1 : 1) * y[i] * A / B;
30      }
31      return ans;
32  }
33  // find 1^k + 2^k + ... + n^k. in O(k) of time complexity.
34  inline mint sum_of_kth_powers(mint n, int k) {
35      mint sum = 0;
36      std::vector<mint> Y{0};
37      for (int i = 1; i <= k + 2; i++) {
38          Y.push_back(sum += (mint)i ^ k);
39      }
40      return cont_lagrange(Y, n);
41  }
```

3.5 光速幂

Listing 28: **speed-of-light-power.cpp**

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

template <int base, int mod>
struct SOLPower { // Speed Of Light Power.
    // p1 stores base^0  ~ base^sq
    // p2 stores base^sq ~ base^(sq^2)
    std::vector<int> p1, ps;
    static const int sq = std::sqrt(mod);
    SOLPower(void) {
        p1.push_back(1), ps.push_back(1);
        for (int i = 1; i <= sq; i++) {
            p1.push_back(1ll * p1.back() * base % mod);
        }
        ps.push_back(p1.back());
        for (int i = 2 * sq; i <= mod; i += sq) {
            ps.push_back(1ll * ps.back() * ps[1] % mod);
        }
    }
    inline int power(int index) {
#if __cplusplus >= 202002L
        if (index > mod) [[unlikely]] {
            index %= mod;
        }
#else
        if (index > mod) index %= mod;
#endif
        return 1ll * ps[index / sq] * p1[index % sq] % mod;
    }
};
```

4 字符串算法

4.1 字符串哈希

Listing 29: **hashed-string.cpp**

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

template <int mod, int seed>
struct SingleHash {
    int n;
    std::vector<int> pow, h;
    SingleHash(void) = default;
    SingleHash(std::string &s) { init(s); }
    inline void init(std::string &s) {
        n = s.size(), h.assign(n + 2, 0), pow.assign(n + 2, 1);
        for (int i = 1; i <= n; i++) {
            pow[i] = 1ll * pow[i - 1] * seed % mod;
            h[i] = (1ll * h[i - 1] * seed + s[i - 1]) % mod;
        }
    }
    inline int get_hash(int l, int r) {
        return (h[r + 1] - 1ll * h[l] * pow[r - l + 1] % mod + mod) % mod;
    }
    inline bool check_same(int l1, int r1, int l2, int r2) {
        return get_hash(l1, r1) == get_hash(l2, r2);
    }
};
struct HashedString {
```

```
    SingleHash<998244353, 477> H1;
    SingleHash<1000000007, 233> H2;
    HashedString(void) = default;
    HashedString(std::string s) : H1(s), H2(s) {}
    inline void init(std::string s) {
        H1.init(s), H2.init(s);
    }
    std::pair<int, int> get_hash(int l, int r) { // not recommended.
        return {H1.get_hash(l, r), H2.get_hash(l, r)};
    }
    // caution: index begins with zero.
    // If index beginning with one is wanted, use s = ' ' + s
    inline bool check_same(int l1, int r1, int l2, int r2) {
        return H1.check_same(l1, r1, l2, r2) && H2.check_same(l1, r1, l2, r2);
    }
    inline bool check_period(int l, int r, int p) {
        return check_same(l, r - p, l + p, r);
    }
};
```

5 图论

5.1 模块化基本图

Listing 30: **graph.cpp**

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

struct NT {}; // null_type
template <class W = NT>
class Graph {
private:
    struct Edge {
        int to;
        W w;
    };
    std::vector<std::vector<Edge>> G; // (to, E)

public:
    Graph(void) {}
    Graph(int n) : G(n + 1) {}
    inline void clear(void) { G.clear(); }
    inline void resize(int n) { G.resize(n + 1); }
    std::vector<Edge> &operator[](int x) { return G[x]; }
    inline void add_directed(int u, int v, W w = W{}) {
        G[u].push_back({v, w});
    }
    inline void add_undirected(int u, int v, W w = W{}) {
        G[u].push_back({v, w});
        G[v].push_back({u, w});
    }
    std::vector<W> dijkstra(int s) { // by default the shortest path.
        using Node = std::pair<W, int>;
        std::vector<W> dis(G.size(), std::numeric_limits<W>::max());
        std::priority_queue<Node, std::vector<Node>, std::greater<Node>> heap;
        heap.push({W{}, s});
        while (!heap.empty()) {
            auto t = heap.top();
            heap.pop();
            if (dis[t.second] != t.first) {
                continue;
            }
```

```
            for (auto &edge : G[t.second]) {
                if (dis[edge.to] > t.first + edge.w) {
                    dis[edge.to] = t.first + edge.w;
                    heap.push({dis[edge.to], edge.to});
                }
            }
        }
        return dis;
    }
};
```

5.2 O(1) LCA

Listing 31: **lca.cpp**

```
#include "sparse-table.h"
#include <bits/stdc++.h>

std::vector<int> G[100010]; // requires a previous graph definition.

class LCAImpl {
private:
    std::vector<int> dfn, seq; // dfn and seq are (internally) zero indexed.
    static std::vector<int> d;
    struct EulerTourInfo {
        int val;
        EulerTourInfo(void) : val(0) {}
        EulerTourInfo(int x) : val(x) {}
        EulerTourInfo operator+(EulerTourInfo &x) {
            return d[val] < d[x.val] ? val : x.val;
        }
    };
    SparseTable<EulerTourInfo> lca; // 0 indexed.
    inline void _dfs(int x, int fa) {
        dfn[x] = seq.size();
        seq.push_back(x), d[x] = d[fa] + 1;
        for (auto &y : G[x]) {
            if (y == fa) continue;
            _dfs(y, x), seq.push_back(x);
        }
    }

public:
    inline void init(int n) {
        d.assign(n + 1, 0), dfn.assign(n + 1, 0);
        seq.clear(), _dfs(1, 0), lca.init(seq);
    }
    inline int LCA(int u, int v) {
        if (u == 0 || v == 0) return u | v;
        if (dfn[u] > dfn[v]) std::swap(u, v);
        return lca.query(dfn[u], dfn[v]).val;
    }
} LCA;
```

6 jiangly 代码库 (备用，侵权请提出 issue)

6.1 01 - int128 库函数自定义

Listing 32: **others/01-i128-Func.cpp**

```
/**   int128 输出流自定义
 *    2023-03-20: https://codeforces.com/contest/1806/submission/198413531
```

```
 **/
using i128 = __int128;

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;
}

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

i128 toi128(const std::string &s) {
    i128 n = 0;
    for (auto c : s) {
        n = n * 10 + (c - '0');
    }
    return n;
}

i128 sqrti128(i128 n) {
    i128 lo = 0, hi = 1E16;
    while (lo < hi) {
        i128 x = (lo + hi + 1) / 2;
        if (x * x <= n) {
            lo = x;
        } else {
            hi = x - 1;
        }
    }
    return lo;
}
```

6.2 02 - 常用库函数重载

Listing 33: **others/02-Math-Func.cpp**

```
using i64 = long long;
using i128 = __int128;

/**   上取整下取整
 *    2023-10-15: https://codeforces.com/contest/293/submission/228297248
**/
i64 ceilDiv(i64 n, i64 m) {
    if (n >= 0) {
        return (n + m - 1) / m;
    } else {
        return n / m;
    }
}

i64 floorDiv(i64 n, i64 m) {
    if (n >= 0) {
        return n / m;
```

```
    } else {
        return (n - m + 1) / m;
    }
}

/**  最大值赋值
 *   2023-09-30: https://codeforces.com/contest/1874/submission/226069129
**/
template<class T>
void chmax(T &a, T b) {
    if (a < b) {
        a = b;
    }
}

/**  最大公约数
 *   -: -
**/
i128 gcd(i128 a, i128 b) {
    return b ? gcd(b, a % b) : a;
}

/**  精确开平方
 *   2024-03-02: https://qoj.ac/submission/343317
**/
i64 sqrt(i64 n) {
    i64 s = std::sqrt(n);
    while (s * s > n) {
        s--;
    }
    while ((s + 1) * (s + 1) <= n) {
        s++;
    }
    return s;
}

/**  精确开平方
 *   2023-09-19: https://qoj.ac/submission/183430
**/
i64 get(i64 n) {
    i64 u = std::sqrt(2.0L * n);
    while (u * (u + 1) / 2 < n) {
        u++;
    }
    while (u * (u - 1) / 2 + 1 > n) {
        u--;
    }
    return u;
}
```

6.3 03 - 字符调整

Listing 34: **others/03-Char.cpp**

```
/**  大小写转换、获取字母序
 *   2024-03-16: https://qoj.ac/submission/355156
**/
void rev(std::string &s) {
    int l = s.size();
    for (int i = 1; i < l; i += 2) {
        if (std::isupper(s[i])) {
            s[i] = std::tolower(s[i]);
        } else {
            s[i] = std::toupper(s[i]);
```

```
        }
    }
}

int get(char c) {
    int x;
    if (std::islower(c)) {
        x = c - 'a';
    } else {
        x = 26 + c - 'A';
    }
    return x;
}
```

6.4 04A - 二分算法（整数域）

Listing 35: **others/04A-Binary-Search.cpp**

```
/**  二分算法（整数域）：前驱
 *   2023-09-18: https://qoj.ac/submission/182628
**/
int lo = 1, hi = 1E9;
while (lo < hi) {
    int m = (lo + hi + 1) / 2;
    if (check(m)) {
        lo = m;
    } else {
        hi = m - 1;
    }
}
std::cout << lo << "\n";

/**  二分算法（整数域）：后继
 *   2023-09-18: https://qoj.ac/submission/182752
**/
int lo = 1, hi = n;
while (lo < hi) {
    int m = (lo + hi) / 2;
    if (check(m)) {
        hi = m;
    } else {
        lo = m + 1;
    }
}
std::cout << lo << "\n";
```

6.5 04B - 二分算法（实数域）

Listing 36: **others/04B-Binary-Search.cpp**

```
/**  二分算法（实数域）
 *   2023-10-21: https://qoj.ac/submission/222042
**/
auto check = [&](double t) {
    // write
};

double lo = 0;
double hi = 1E12;
while (hi - lo > std::max(1.0, lo) * eps) {
    double x = (lo + hi) / 2;
    if (check(x)) {
        hi = x;
    } else {
```

```
        lo = x;
    }
}

std::cout << lo << "\n";

/**  二分算法（实数域）
 *   2023-09-15: https://qoj.ac/submission/179994
**/
using i64 = long long;
using real = long double;

constexpr real eps = 1E-7;

auto get = [&](const auto &f) {
    real lo = -1E4, hi = 1E4;
    while (hi - lo > 3 * eps) {
        real x1 = (lo + hi - eps) / 2;
        real x2 = (lo + hi + eps) / 2;
        if (f(x1) > f(x2)) {
            lo = x1;
        } else {
            hi = x2;
        }
    }
    return f((lo + hi) / 2);
};

std::cout << get([&](real px) {
    return get([&](real py) {
        // write
    });
}) << "\n";
```

6.6 01 - 强连通分量缩点（SCC）

Listing 37: **graph/01-SCC.cpp**

```
/**  强连通分量缩点（SCC）
 *   2023-06-18: https://codeforces.com/contest/1835/submission/210147209
**/
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) {
```

```
27          adj[u].push_back(v);
28      }
29
30      void dfs(int x) {
31          dfn[x] = low[x] = cur++;
32          stk.push_back(x);
33
34          for (auto y : adj[x]) {
35              if (dfn[y] == -1) {
36                  dfs(y);
37                  low[x] = std::min(low[x], low[y]);
38              } else if (bel[y] == -1) {
39                  low[x] = std::min(low[x], dfn[y]);
40              }
41          }
42
43          if (dfn[x] == low[x]) {
44              int y;
45              do {
46                  y = stk.back();
47                  bel[y] = cnt;
48                  stk.pop_back();
49              } while (y != x);
50              cnt++;
51          }
52      }
53
54      std::vector<int> work() {
55          for (int i = 0; i < n; i++) {
56              if (dfn[i] == -1) {
57                  dfs(i);
58              }
59          }
60          return bel;
61      }
62  };
```

6.7 02 - 割边与割边缩点（EBCC）

Listing 38: **graph/02-EBCC.cpp**

```
1   /**   割边与割边缩点（EBCC）
2    *    2023-05-11: https://codeforces.com/contest/118/submission/205426518
3   **/
4   std::set<std::pair<int, int>> E;
5
6   struct EBCC {
7       int n;
8       std::vector<std::vector<int>> adj;
9       std::vector<int> stk;
10      std::vector<int> dfn, low, bel;
11      int cur, cnt;
12
13      EBCC() {}
14      EBCC(int n) {
15          init(n);
16      }
17
18      void init(int n) {
19          this->n = n;
20          adj.assign(n, {});
21          dfn.assign(n, -1);
22          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;
    }
};
```

6.8 03 - 二分图最大权匹配（MaxAssignment 基于 KM）

Listing 39: **graph/03-Max-Assignment.cpp**

```cpp
/**  二分图最大权匹配（MaxAssignment 基于KM）
 *   2022-04-10: https://atcoder.jp/contests/abc247/submissions/30867023
 *   2023-09-21: https://qoj.ac/submission/184824
**/
constexpr int inf = 1E7;
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;
};
```

6.9 04 - 一般图最大匹配（Graph 带花树算法）【久远】

Listing 40: **graph/04-Graph-Match.cpp**

```
/**   一般图最大匹配（Graph 带花树算法）【久远】
 *    2021-12-24: https://codeforces.com/contest/1615/submission/140509278
**/
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;
                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)
```

```
48          auto augment = [&](int u) {
49
50              while (!que.empty())
51                  que.pop();
52
53              std::iota(f.begin(), f.end(), 0);
54
55              // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
56              std::fill(vis.begin(), vis.end(), -1);
57
58              que.push(u);
59              vis[u] = 1;
60              dep[u] = 0;
61
62              while (!que.empty()){
63                  int u = que.front();
64                  que.pop();
65                  for (auto v : e[u]) {
66                      if (vis[v] == -1) {
67
68                          vis[v] = 0;
69                          link[v] = u;
70                          dep[v] = dep[u] + 1;
71
72                          // found an augmenting path
73                          if (match[v] == -1) {
74                              for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
75                                  temp = match[y];
76                                  match[x] = y;
77                                  match[y] = x;
78                              }
79                              return;
80                          }
81
82                          vis[match[v]] = 1;
83                          dep[match[v]] = dep[u] + 2;
84                          que.push(match[v]);
85
86                      } else if (vis[v] == 1 && find(v) != find(u)) {
87                          // found a blossom
88                          int p = lca(u, v);
89                          blossom(u, v, p);
90                          blossom(v, u, p);
91                      }
92                  }
93              }
94
95          };
96
97          // find a maximal matching greedily (decrease constant)
98          auto greedy = [&]() {
99
100             for (int u = 0; u < n; ++u) {
101                 if (match[u] != -1)
102                     continue;
103                 for (auto v : e[u]) {
104                     if (match[v] == -1) {
105                         match[u] = v;
106                         match[v] = u;
107                         break;
108                     }
109                 }
110             }
111         };
```

```
112
113         greedy();
114
115         for (int u = 0; u < n; ++u)
116             if (match[u] == -1)
117                 augment(u);
118
119         return match;
120     }
121 };
```

6.10 05 - TwoSat（2-Sat）

Listing 41: **graph/05-Two-Sat.cpp**

```
1  /**  TwoSat（2-Sat）
2   *   2023-09-29: https://atcoder.jp/contests/arc161/submissions/46031530
3  **/
4  struct TwoSat {
5      int n;
6      std::vector<std::vector<int>> e;
7      std::vector<bool> ans;
8      TwoSat(int n) : n(n), e(2 * n), ans(n) {}
9      void addClause(int u, bool f, int v, bool g) {
10         e[2 * u + !f].push_back(2 * v + g);
11         e[2 * v + !g].push_back(2 * u + f);
12     }
13     bool satisfiable() {
14         std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
15         std::vector<int> stk;
16         int now = 0, cnt = 0;
17         std::function<void(int)> tarjan = [&](int u) {
18             stk.push_back(u);
19             dfn[u] = low[u] = now++;
20             for (auto v : e[u]) {
21                 if (dfn[v] == -1) {
22                     tarjan(v);
23                     low[u] = std::min(low[u], low[v]);
24                 } else if (id[v] == -1) {
25                     low[u] = std::min(low[u], dfn[v]);
26                 }
27             }
28             if (dfn[u] == low[u]) {
29                 int v;
30                 do {
31                     v = stk.back();
32                     stk.pop_back();
33                     id[v] = cnt;
34                 } while (v != u);
35                 ++cnt;
36             }
37         };
38         for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
39         for (int i = 0; i < n; ++i) {
40             if (id[2 * i] == id[2 * i + 1]) return false;
41             ans[i] = id[2 * i] > id[2 * i + 1];
42         }
43         return true;
44     }
45     std::vector<bool> answer() { return ans; }
46 };
```

6.11 06A - 最大流（Flow 旧版其一，整数应用）

Listing 42: **graph/06A-Max-Flow.cpp**

```cpp
/**  最大流（Flow 旧版其一，整数应用）
 *   2022-09-03: https://codeforces.com/contest/1717/submission/170688062
**/
template<class T>
struct Flow {
    const int n;
    struct Edge {
        int to;
        T cap;
        Edge(int to, T cap) : to(to), cap(cap) {}
    };
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> cur, h;
    Flow(int n) : n(n), g(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);
```

```
62          g[v].push_back(e.size());
63          e.emplace_back(u, 0);
64      }
65      T maxFlow(int s, int t) {
66          T ans = 0;
67          while (bfs(s, t)) {
68              cur.assign(n, 0);
69              ans += dfs(s, t, std::numeric_limits<T>::max());
70          }
71          return ans;
72      }
73  };
```

6.12 06B - 最大流（Flow 旧版其二，浮点数应用）

Listing 43: **graph/06B-Max-Flow.cpp**

```
1   /**   最大流（Flow 旧版其二，浮点数应用）
2    *    2022-04-09: https://cf.dianhsu.com/gym/104288/submission/201412765
3   **/
4   template<class T>
5   struct Flow {
6       const int n;
7       struct Edge {
8           int to;
9           T cap;
10          Edge(int to, T cap) : to(to), cap(cap) {}
11      };
12      std::vector<Edge> e;
13      std::vector<std::vector<int>> g;
14      std::vector<int> cur, h;
15      Flow(int n) : n(n), g(n) {}
16
17      bool bfs(int s, int t) {
18          h.assign(n, -1);
19          std::queue<int> que;
20          h[s] = 0;
21          que.push(s);
22          while (!que.empty()) {
23              const int u = que.front();
24              que.pop();
25              for (int i : g[u]) {
26                  auto [v, c] = e[i];
27                  if (c > 0 && h[v] == -1) {
28                      h[v] = h[u] + 1;
29                      if (v == t) {
30                          return true;
31                      }
32                      que.push(v);
33                  }
34              }
35          }
36          return false;
37      }
38
39      T dfs(int u, int t, T f) {
40          if (u == t) {
41              return f;
42          }
43          auto r = f;
44          double res = 0;
45          for (int &i = cur[u]; i < int(g[u].size()); ++i) {
46              const int j = g[u][i];
47              auto [v, c] = e[j];
```

```
48              if (c > 0 && h[v] == h[u] + 1) {
49                  auto a = dfs(v, t, std::min(r, c));
50                  res += a;
51                  e[j].cap -= a;
52                  e[j ^ 1].cap += a;
53                  r -= a;
54                  if (r == 0) {
55                      return f;
56                  }
57              }
58          }
59          return res;
60      }
61      void addEdge(int u, int v, T c) {
62          g[u].push_back(e.size());
63          e.emplace_back(v, c);
64          g[v].push_back(e.size());
65          e.emplace_back(u, 0);
66      }
67      T maxFlow(int s, int t) {
68          T ans = 0;
69          while (bfs(s, t)) {
70              cur.assign(n, 0);
71              ans += dfs(s, t, 1E100);
72          }
73          return ans;
74      }
75  };
```

6.13 06C - 最大流（MaxFlow 新版）

Listing 44: **graph/06C-Max-Flow.cpp**

```
1   /**   最大流（MaxFlow 新版）
2    *    2023-07-21: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=62915815
3   **/
4   constexpr int inf = 1E9;
5   template<class T>
6   struct MaxFlow {
7       struct _Edge {
8           int to;
9           T cap;
10          _Edge(int to, T cap) : to(to), cap(cap) {}
11      };
12
13      int n;
14      std::vector<_Edge> e;
15      std::vector<std::vector<int>> g;
16      std::vector<int> cur, h;
17
18      MaxFlow() {}
19      MaxFlow(int n) {
20          init(n);
21      }
22
23      void init(int n) {
24          this->n = n;
25          e.clear();
26          g.assign(n, {});
27          cur.resize(n);
28          h.resize(n);
29      }
30
31      bool bfs(int s, int t) {
```

```
32          h.assign(n, -1);
33          std::queue<int> que;
34          h[s] = 0;
35          que.push(s);
36          while (!que.empty()) {
37              const int u = que.front();
38              que.pop();
39              for (int i : g[u]) {
40                  auto [v, c] = e[i];
41                  if (c > 0 && h[v] == -1) {
42                      h[v] = h[u] + 1;
43                      if (v == t) {
44                          return true;
45                      }
46                      que.push(v);
47                  }
48              }
49          }
50          return false;
51      }
52
53      T dfs(int u, int t, T f) {
54          if (u == t) {
55              return f;
56          }
57          auto r = f;
58          for (int &i = cur[u]; i < int(g[u].size()); ++i) {
59              const int j = g[u][i];
60              auto [v, c] = e[j];
61              if (c > 0 && h[v] == h[u] + 1) {
62                  auto a = dfs(v, t, std::min(r, c));
63                  e[j].cap -= a;
64                  e[j ^ 1].cap += a;
65                  r -= a;
66                  if (r == 0) {
67                      return f;
68                  }
69              }
70          }
71          return f - r;
72      }
73      void addEdge(int u, int v, T c) {
74          g[u].push_back(e.size());
75          e.emplace_back(v, c);
76          g[v].push_back(e.size());
77          e.emplace_back(u, 0);
78      }
79      T flow(int s, int t) {
80          T ans = 0;
81          while (bfs(s, t)) {
82              cur.assign(n, 0);
83              ans += dfs(s, t, std::numeric_limits<T>::max());
84          }
85          return ans;
86      }
87
88      std::vector<bool> minCut() {
89          std::vector<bool> c(n);
90          for (int i = 0; i < n; i++) {
91              c[i] = (h[i] != -1);
92          }
93          return c;
94      }
```

```
95
96      struct Edge {
97          int from;
98          int to;
99          T cap;
100         T flow;
101     };
102     std::vector<Edge> edges() {
103         std::vector<Edge> a;
104         for (int i = 0; i < e.size(); i += 2) {
105             Edge x;
106             x.from = e[i + 1].to;
107             x.to = e[i].to;
108             x.cap = e[i].cap + e[i + 1].cap;
109             x.flow = e[i + 1].cap;
110             a.push_back(x);
111         }
112         return a;
113     }
114 };
```

6.14 07A - 费用流（MCFGraph 旧版）

Listing 45: **graph/07A-Min-Cost-Flow.cpp**

```
1  /**   费用流（MCFGraph 旧版）
2   *    2022-12-12: https://codeforces.com/contest/1766/submission/184974697
3   *
4   *    下方为最小费用**最大流**模板，如需求解最小费用**可行流**，需要去除建边限制
5  **/
6  struct MCFGraph {
7      struct Edge {
8          int v, c, f;
9          Edge(int v, int c, int f) : v(v), c(c), f(f) {}
10     };
11     const int n;
12     std::vector<Edge> e;
13     std::vector<std::vector<int>> g;
14     std::vector<i64> h, dis;
15     std::vector<int> pre;
16     bool dijkstra(int s, int t) {
17         dis.assign(n, std::numeric_limits<i64>::max());
18         pre.assign(n, -1);
19         std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>, std::greater<std::pair<
             i64, int>>> que;
20         dis[s] = 0;
21         que.emplace(0, s);
22         while (!que.empty()) {
23             i64 d = que.top().first;
24             int u = que.top().second;
25             que.pop();
26             if (dis[u] < d) continue;
27             for (int i : g[u]) {
28                 int v = e[i].v;
29                 int c = e[i].c;
30                 int f = e[i].f;
31                 if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
32                     dis[v] = d + h[u] - h[v] + f;
33                     pre[v] = i;
34                     que.emplace(dis[v], v);
35                 }
36             }
37         }
38         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) {
        // 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);
        // }
    }
    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);
    }
};
```

6.15 07B - 费用流（MinCostFlow 新版）

Listing 46: **graph/07B-Min-Cost-Flow.cpp**

```
/**  MinCostFlow 新版
 *   2023-11-09: https://qoj.ac/submission/244680
**/
template<class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
    };
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T,
          int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T 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].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) {
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::make_pair(flow, cost);
    }
    struct Edge {
        int from;
        int to;
        T cap;
        T cost;
        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.cost = e[i].cost;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

6.16 08 - 树链剖分（HLD）

Listing 47: **graph/08-HLD.cpp**

```
/**   树链剖分（HLD）
 *    2023-08-31: https://codeforces.com/contest/1863/submission/221214363
**/
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);
```

```
47              siz[u] += siz[v];
48              if (siz[v] > siz[adj[u][0]]) {
49                  std::swap(v, adj[u][0]);
50              }
51          }
52      }
53      void dfs2(int u) {
54          in[u] = cur++;
55          seq[in[u]] = u;
56          for (auto v : adj[u]) {
57              top[v] = v == adj[u][0] ? top[u] : v;
58              dfs2(v);
59          }
60          out[u] = cur;
61      }
62      int lca(int u, int v) {
63          while (top[u] != top[v]) {
64              if (dep[top[u]] > dep[top[v]]) {
65                  u = parent[top[u]];
66              } else {
67                  v = parent[top[v]];
68              }
69          }
70          return dep[u] < dep[v] ? u : v;
71      }
72
73      int dist(int u, int v) {
74          return dep[u] + dep[v] - 2 * dep[lca(u, v)];
75      }
76
77      int jump(int u, int k) {
78          if (dep[u] < k) {
79              return -1;
80          }
81
82          int d = dep[u] - k;
83
84          while (dep[top[u]] > d) {
85              u = parent[top[u]];
86          }
87
88          return seq[in[u] - dep[u] + d];
89      }
90
91      bool isAncester(int u, int v) {
92          return in[u] <= in[v] && in[v] < out[u];
93      }
94
95      int rootedParent(int u, int v) {
96          std::swap(u, v);
97          if (u == v) {
98              return u;
99          }
100         if (!isAncester(u, v)) {
101             return parent[u];
102         }
103         auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
104             return in[x] < in[y];
105         }) - 1;
106         return *it;
107     }
108
109     int rootedSize(int u, int v) {
110         if (u == v) {
```

```
            return n;
        }
        if (!isAncester(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);
    }
};
```

6.17 01 - 快速幂

Listing 48: **math/01-Power.cpp**

```
/**   快速幂 - 普通版
 *    2023-10-09: https://atcoder.jp/contests/tenka1-2017/submissions/46411797
**/
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;
}

/**   快速幂 - 手写乘法
 *    2023-09-27: https://qoj.ac/submission/189343
**/
using i64 = long long;

i64 mul(i64 a, i64 b, i64 p) {
    i64 c = a * b - i64(1.0L * a * b / p) * p;
    c %= p;
    if (c < 0) {
        c += p;
    }
    return c;
}

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

6.18 02 - 基姆拉尔森公式

Listing 49: **math/02-Kim-Larsen.cpp**

```
/**   基姆拉尔森公式
 *    2023-09-05: https://qoj.ac/submission/164735
**/
const int d[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

```

```
bool isLeap(int y) {
    return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
}

int daysInMonth(int y, int m) {
    return d[m - 1] + (isLeap(y) && m == 2);
}

int getDay(int y, int m, int d) {
    int ans = 0;
    for (int i = 1970; i < y; i++) {
        ans += 365 + isLeap(i);
    }
    for (int i = 1; i < m; i++) {
        ans += daysInMonth(y, i);
    }
    ans += d;
    return (ans + 2) % 7 + 1;
}
```

6.19 03 - 欧拉筛

Listing 50: **math/03-Euler-Sieve.cpp**

```
/**   欧拉筛
 *    2023-11-14: https://qoj.ac/submission/251234
**/
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;
            }
        }
    }
}

bool isprime(int n) {
    return minp[n] == n;
}
```

6.20 04 - 莫比乌斯函数筛（莫比乌斯反演）

Listing 51: **math/04-Mu-Sieve.cpp**

```
/**   莫比乌斯函数筛（莫比乌斯函数/反演）
 *    2023-03-04: https://atcoder.jp/contests/tupc2022/submissions/39391116
**/
std::unordered_map<int, Z> fMu;
```

```
5
6  constexpr int N = 1E7;
7  std::vector<int> minp, primes;
8  std::vector<Z> mu;
9
10 void sieve(int n) {
11     minp.assign(n + 1, 0);
12     mu.resize(n);
13     primes.clear();
14
15     mu[1] = 1;
16     for (int i = 2; i <= n; i++) {
17         if (minp[i] == 0) {
18             mu[i] = -1;
19             minp[i] = i;
20             primes.push_back(i);
21         }
22
23         for (auto p : primes) {
24             if (i * p > n) {
25                 break;
26             }
27             minp[i * p] = p;
28             if (p == minp[i]) {
29                 break;
30             }
31             mu[i * p] = -mu[i];
32         }
33     }
34
35     for (int i = 1; i <= n; i++) {
36         mu[i] += mu[i - 1];
37     }
38 }
39
40 Z sumMu(int n) {
41     if (n <= N) {
42         return mu[n];
43     }
44     if (fMu.count(n)) {
45         return fMu[n];
46     }
47     if (n == 0) {
48         return 0;
49     }
50     Z ans = 1;
51     for (int l = 2, r; l <= n; l = r + 1) {
52         r = n / (n / l);
53         ans -= (r - l + 1) * sumMu(n / l);
54     }
55     return ans;
56 }
```

6.21 05 - 扩展欧几里得（exgcd）

Listing 52: **math/05-Exgcd.cpp**

```
1  /**   扩展欧几里得（exgcd）
2   *    2023-10-09: https://atcoder.jp/contests/tenka1-2017/submissions/46411797
3  **/
4  int exgcd(int a, int b, int &x, int &y) {
5      if (!b) {
6          x = 1, y = 0;
7          return a;
```

```
    }
    int g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}

/**  扩展欧几里得（exgcd）
 *   2023-09-05: https://qoj.ac/submission/165983
**/
std::array<i64, 3> exgcd(i64 a, i64 b) {
    if (!b) {
        return {a, 1, 0};
    }
    auto [g, x, y] = exgcd(b, a % b);
    return {g, y, x - a / b * y};
}
```

6.22 06A - 欧拉函数（求解单个数的欧拉函数）

Listing 53: **math/06A-Phi.cpp**

```
/**  欧拉函数（求解单个数的欧拉函数）
 *   2023-10-09: https://atcoder.jp/contests/tenka1-2017/submissions/46411797
**/
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;
}
```

6.23 06B - 欧拉函数（求解全部数的欧拉函数）

Listing 54: **math/06B-Phi-Sieve.cpp**

```
/**  欧拉函数（求解全部数的欧拉函数）
 *   2023-09-24: https://qoj.ac/submission/187055
 **/
constexpr int N = 1E7;
constexpr int P = 1000003;

bool isprime[N + 1];
int phi[N + 1];
std::vector<int> primes;

void sieve(void) {
    std::fill(isprime + 2, isprime + N + 1, true);
    phi[1] = 1;
    for (int i = 2; i <= N; i++) {
        if (isprime[i]) {
            primes.push_back(i);
            phi[i] = i - 1;
        }
        for (auto p : primes) {
```

```
            if (i * p > N) {
                break;
            }
            isprime[i * p] = false;
            if (i % p == 0) {
                phi[i * p] = phi[i] * p;
                break;
            }
            phi[i * p] = phi[i] * (p - 1);
        }
    }
}
```

6.24 07A - 组合数（小范围预处理，逆元 + 杨辉三角）

Listing 55: **math/07A-Comb.cpp**

```
/**  组合数（小范围预处理，逆元+杨辉三角）
 *   2024-03-14: https://qoj.ac/submission/353877
 *   2023-10-06: https://qoj.ac/submission/203196
**/
constexpr int P = 1000000007;
constexpr int L = 10000;

int fac[L + 1], invfac[L + 1];
int sumbinom[L + 1][7];

int binom(int n, int m) {
    if (n < m || m < 0) {
        return 0;
    }
    return 1LL * fac[n] * invfac[m] % P * invfac[n - m] % P;
}

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

int main() {
    fac[0] = 1;
    for (int i = 1; i <= L; i++) {
        fac[i] = 1LL * fac[i - 1] * i % P;
    }
    invfac[L] = power(fac[L], P - 2);
    for (int i = L; i; i--) {
        invfac[i - 1] = 1LL * invfac[i] * i % P;
    }

    sumbinom[0][0] = 1;
    for (int i = 1; i <= L; i++) {
        for (int j = 0; j < 7; j++) {
            sumbinom[i][j] = (sumbinom[i - 1][j] + sumbinom[i - 1][(j + 6) % 7]) % P;
        }
    }
}
```

6.25 07B - 组合数（Comb, with. ModIntBase）

Listing 56: **math/07B-Comb.cpp**

```cpp
/**   组合数（Comb, with. MInt & MLong）
 *    2023-08-26: https://codeforces.com/contest/1864/submission/220584872
**/
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;
```

6.26 08 - 素数测试与因式分解（Miller-Rabin and Pollard-Rho）

Listing 57: **math/08-Prime.cpp**

```cpp
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
```

```
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
std::vector<i64> factorize(i64 n) {
    std::vector<i64> p;
    std::function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
        if (isprime(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {
            return (mul(x, x, n) + 1) % n;
        };
        i64 x0 = 2;
        while (true) {
            i64 x = x0;
            i64 y = x0;
            i64 d = 1;
            i64 power = 1, lam = 0;
            i64 v = 1;
            while (d == 1) {
                y = g(y);
                ++lam;
                v = mul(v, std::abs(x - y), n);
                if (lam % 127 == 0) {
                    d = std::gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
```

```
                    lam = 0;
                    d = std::gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    std::sort(p.begin(), p.end());
    return p;
}
```

6.27 09A - 平面几何（Point）

Listing 58: **math/09A-Flat-Geometry.cpp**

```
/**  平面几何（Point）
 *   2023-09-22: https://qoj.ac/submission/185408
**/
template<class T>
struct Point {
    T x;
    T y;
    Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}

    template<class U>
    operator Point<U>() {
        return Point<U>(U(x), U(y));
    }
    Point &operator+=(const Point &p) & {
        x += p.x;
        y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) & {
        x -= p.x;
        y -= p.y;
        return *this;
    }
    Point &operator*=(const T &v) & {
        x *= v;
        y *= v;
        return *this;
    }
    Point &operator/=(const T &v) & {
        x /= v;
        y /= v;
        return *this;
    }
    Point operator-() const {
        return Point(-x, -y);
    }
    friend Point operator+(Point a, const Point &b) {
        return a += b;
    }
    friend Point operator-(Point a, const Point &b) {
        return a -= b;
    }
```

```
43      friend Point operator*(Point a, const T &b) {
44          return a *= b;
45      }
46      friend Point operator/(Point a, const T &b) {
47          return a /= b;
48      }
49      friend Point operator*(const T &a, Point b) {
50          return b *= a;
51      }
52      friend bool operator==(const Point &a, const Point &b) {
53          return a.x == b.x && a.y == b.y;
54      }
55      friend std::istream &operator>>(std::istream &is, Point &p) {
56          return is >> p.x >> p.y;
57      }
58      friend std::ostream &operator<<(std::ostream &os, const Point &p) {
59          return os << "(" << p.x << ",␣" << p.y << ")";
60      }
61  };
62
63  template<class T>
64  struct Line {
65      Point<T> a;
66      Point<T> b;
67      Line(const Point<T> &a_ = Point<T>(), const Point<T> &b_ = Point<T>()) : a(a_), b(b_) {}
68  };
69
70  template<class T>
71  T dot(const Point<T> &a, const Point<T> &b) {
72      return a.x * b.x + a.y * b.y;
73  }
74
75  template<class T>
76  T cross(const Point<T> &a, const Point<T> &b) {
77      return a.x * b.y - a.y * b.x;
78  }
79
80  template<class T>
81  T square(const Point<T> &p) {
82      return dot(p, p);
83  }
84
85  template<class T>
86  double length(const Point<T> &p) {
87      return std::sqrt(square(p));
88  }
89
90  template<class T>
91  double length(const Line<T> &l) {
92      return length(l.a - l.b);
93  }
94
95  template<class T>
96  Point<T> normalize(const Point<T> &p) {
97      return p / length(p);
98  }
99
100 template<class T>
101 bool parallel(const Line<T> &l1, const Line<T> &l2) {
102     return cross(l1.b - l1.a, l2.b - l2.a) == 0;
103 }
104
105 template<class T>
106 double distance(const Point<T> &a, const Point<T> &b) {
```

```
107     return length(a - b);
108 }
109
110 template<class T>
111 double distancePL(const Point<T> &p, const Line<T> &l) {
112     return std::abs(cross(l.a - l.b, l.a - p)) / length(l);
113 }
114
115 template<class T>
116 double distancePS(const Point<T> &p, const Line<T> &l) {
117     if (dot(p - l.a, l.b - l.a) < 0) {
118         return distance(p, l.a);
119     }
120     if (dot(p - l.b, l.a - l.b) < 0) {
121         return distance(p, l.b);
122     }
123     return distancePL(p, l);
124 }
125
126 template<class T>
127 Point<T> rotate(const Point<T> &a) {
128     return Point(-a.y, a.x);
129 }
130
131 template<class T>
132 int sgn(const Point<T> &a) {
133     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
134 }
135
136 template<class T>
137 bool pointOnLineLeft(const Point<T> &p, const Line<T> &l) {
138     return cross(l.b - l.a, p - l.a) > 0;
139 }
140
141 template<class T>
142 Point<T> lineIntersection(const Line<T> &l1, const Line<T> &l2) {
143     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b));
144 }
145
146 template<class T>
147 bool pointOnSegment(const Point<T> &p, const Line<T> &l) {
148     return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x && p.x <= std::max(l.a.x, l.b.x)
149         && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
150 }
151
152 template<class T>
153 bool pointInPolygon(const Point<T> &a, const std::vector<Point<T>> &p) {
154     int n = p.size();
155     for (int i = 0; i < n; i++) {
156         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
157             return true;
158         }
159     }
160
161     int t = 0;
162     for (int i = 0; i < n; i++) {
163         auto u = p[i];
164         auto v = p[(i + 1) % n];
165         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
166             t ^= 1;
167         }
168         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
169             t ^= 1;
```

```
        }
    }

    return t == 1;
}

// 0 : not intersect
// 1 : strictly intersect
// 2 : overlap
// 3 : intersect at endpoint
template<class T>
std::tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1, const Line<T> &l2) {
    if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
        if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
            return {0, Point<T>(), Point<T>()};
        } else {
            auto maxx1 = std::max(l1.a.x, l1.b.x);
            auto minx1 = std::min(l1.a.x, l1.b.x);
            auto maxy1 = std::max(l1.a.y, l1.b.y);
            auto miny1 = std::min(l1.a.y, l1.b.y);
            auto maxx2 = std::max(l2.a.x, l2.b.x);
            auto minx2 = std::min(l2.a.x, l2.b.x);
            auto maxy2 = std::max(l2.a.y, l2.b.y);
            auto miny2 = std::min(l2.a.y, l2.b.y);
            Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
            Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
            if (!pointOnSegment(p1, l1)) {
                std::swap(p1.y, p2.y);
            }
            if (p1 == p2) {
                return {3, p1, p2};
            } else {
                return {2, p1, p2};
            }
        }
    }
    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);

    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0 && cp4 < 0)) {
        return {0, Point<T>(), Point<T>()};
    }

    Point p = lineIntersection(l1, l2);
    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
        return {1, p, p};
    } else {
        return {3, p, p};
    }
```

```
}

template<class T>
double distanceSS(const Line<T> &l1, const Line<T> &l2) {
    if (std::get<0>(segmentIntersection(l1, l2)) != 0) {
        return 0.0;
    }
    return std::min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(l2.a, l1), distancePS(l2.b, l1)
      });
}

template<class T>
bool segmentInPolygon(const Line<T> &l, const std::vector<Point<T>> &p) {
    int n = p.size();
    if (!pointInPolygon(l.a, p)) {
        return false;
    }
    if (!pointInPolygon(l.b, p)) {
        return false;
    }
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        auto w = p[(i + 2) % n];
        auto [t, p1, p2] = segmentIntersection(l, Line(u, v));

        if (t == 1) {
            return false;
        }
        if (t == 0) {
            continue;
        }
        if (t == 2) {
            if (pointOnSegment(v, l) && v != l.a && v != l.b) {
                if (cross(v - u, w - v) > 0) {
                    return false;
                }
            }
        } else {
            if (p1 != u && p1 != v) {
                if (pointOnLineLeft(l.a, Line(v, u))
                    || pointOnLineLeft(l.b, Line(v, u))) {
                    return false;
                }
            } else if (p1 == v) {
                if (l.a == v) {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, l)
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else if (l.b == v) {
                    if (pointOnLineLeft(u, Line(l.b, l.a))) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
```

```
                    } else {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                }
            }
        }
    }
    return true;
}

template<class T>
std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
    std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
        auto d1 = l1.b - l1.a;
        auto d2 = l2.b - l2.a;

        if (sgn(d1) != sgn(d2)) {
            return sgn(d1) == 1;
        }

        return cross(d1, d2) > 0;
    });

    std::deque<Line<T>> ls;
    std::deque<Point<T>> ps;
    for (auto l : lines) {
        if (ls.empty()) {
            ls.push_back(l);
            continue;
        }

        while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
            ps.pop_back();
            ls.pop_back();
        }

        while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
            ps.pop_front();
            ls.pop_front();
        }

        if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
            if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {

                if (!pointOnLineLeft(ls.back().a, l)) {
                    assert(ls.size() == 1);
                    ls[0] = l;
                }
                continue;
```

```
            }
            return {};
        }

        ps.push_back(lineIntersection(ls.back(), l));
        ls.push_back(l);
    }

    while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
        ps.pop_back();
        ls.pop_back();
    }
    if (ls.size() <= 2) {
        return {};
    }
    ps.push_back(lineIntersection(ls[0], ls.back()));

    return std::vector(ps.begin(), ps.end());
}

using real = long double;
using P = Point<real>;

constexpr real eps = 0;
```

6.28 09B - 平面几何（with. std::complex）

Listing 59: **math/09B-Flat-Geometry.cpp**

```
/**   平面几何（with. std::complex）
 *    2023-09-04: https://qoj.ac/submission/164445
**/
using Point = std::complex<long double>;

#define x real
#define y imag

long double dot(const Point &a, const Point &b) {
    return (std::conj(a) * b).x();
}

long double cross(const Point &a, const Point &b) {
    return (std::conj(a) * b).y();
}

long double length(const Point &a) {
    return std::sqrt(dot(a, a));
}

long double dist(const Point &a, const Point &b) {
    return length(a - b);
}

long double get(const Point &a, const Point &b, const Point &c, const Point &d) {
    auto e = a + (b - a) * cross(c - a, d - a) / cross(b - a, d - c);
    return dist(d, e);
}
```

6.29 10 - 立体几何（Point）

Listing 60: **math/10-Solid-Geometry.cpp**

```
/**   立体几何
```

```
 *    2023-09-25 (i64): https://qoj.ac/submission/188519
 *    2023-09-28 (double): https://qoj.ac/submission/190463
**/
using i64 = long long;
using real = double;

struct Point {
    real x = 0;
    real y = 0;
    real z = 0;
};

Point operator+(const Point &a, const Point &b) {
    return {a.x + b.x, a.y + b.y, a.z + b.z};
}

Point operator-(const Point &a, const Point &b) {
    return {a.x - b.x, a.y - b.y, a.z - b.z};
}

Point operator*(const Point &a, real b) {
    return {a.x * b, a.y * b, a.z * b};
}

Point operator/(const Point &a, real b) {
    return {a.x / b, a.y / b, a.z / b};
}

real length(const Point &a) {
    return std::hypot(a.x, a.y, a.z);
}

Point normalize(const Point &a) {
    real l = length(a);
    return {a.x / l, a.y / l, a.z / l};
}

real getAng(real a, real b, real c) {
    return std::acos((a * a + b * b - c * c) / 2 / a / b);
}

std::ostream &operator<<(std::ostream &os, const Point &a) {
    return os << "(" << a.x << ",_" << a.y << ",_" << a.z << ")";
}

real dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

Point cross(const Point &a, const Point &b) {
    return {
        a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x
    };
}
```

6.30 11A - 静态凸包（with. Point，旧版）

Listing 61: **math/11A-Convex-Hull.cpp**

```
/**   静态凸包（with. Point，旧版）
 *    2023-04-09: https://cf.dianhsu.com/gym/104288/submission/201412835
**/
```

```
4  struct Point {
5      i64 x;
6      i64 y;
7      Point(i64 x = 0, i64 y = 0) : x(x), y(y) {}
8  };
9
10 bool operator==(const Point &a, const Point &b) {
11     return a.x == b.x && a.y == b.y;
12 }
13
14 Point operator+(const Point &a, const Point &b) {
15     return Point(a.x + b.x, a.y + b.y);
16 }
17
18 Point operator-(const Point &a, const Point &b) {
19     return Point(a.x - b.x, a.y - b.y);
20 }
21
22 i64 dot(const Point &a, const Point &b) {
23     return a.x * b.x + a.y * b.y;
24 }
25
26 i64 cross(const Point &a, const Point &b) {
27     return a.x * b.y - a.y * b.x;
28 }
29
30 void norm(std::vector<Point> &h) {
31     int i = 0;
32     for (int j = 0; j < int(h.size()); j++) {
33         if (h[j].y < h[i].y || (h[j].y == h[i].y && h[j].x < h[i].x)) {
34             i = j;
35         }
36     }
37     std::rotate(h.begin(), h.begin() + i, h.end());
38 }
39
40 int sgn(const Point &a) {
41     return a.y > 0 || (a.y == 0 && a.x > 0) ? 0 : 1;
42 }
43
44 std::vector<Point> getHull(std::vector<Point> p) {
45     std::vector<Point> h, l;
46     std::sort(p.begin(), p.end(), [&](auto a, auto b) {
47         if (a.x != b.x) {
48             return a.x < b.x;
49         } else {
50             return a.y < b.y;
51         }
52     });
53     p.erase(std::unique(p.begin(), p.end()), p.end());
54     if (p.size() <= 1) {
55         return p;
56     }
57
58     for (auto a : p) {
59         while (h.size() > 1 && cross(a - h.back(), a - h[h.size() - 2]) <= 0) {
60             h.pop_back();
61         }
62         while (l.size() > 1 && cross(a - l.back(), a - l[l.size() - 2]) >= 0) {
63             l.pop_back();
64         }
65         l.push_back(a);
66         h.push_back(a);
67     }
```

```
68
69      l.pop_back();
70      std::reverse(h.begin(), h.end());
71      h.pop_back();
72      l.insert(l.end(), h.begin(), h.end());
73      return l;
74  }
```

6.31 11B - 静态凸包（with. Point，新版）

Listing 62: **math/11B-Convex-Hull.cpp**

```
1   /**   静态凸包（with. Point，新版）
2    *    2024-04-06: https://qoj.ac/submission/379920)
3   **/
4   struct Point {
5       i64 x;
6       i64 y;
7       Point() : x{0}, y{0} {}
8       Point(i64 x_, i64 y_) : x{x_}, y{y_} {}
9   };
10
11  i64 dot(Point a, Point b) {
12      return a.x * b.x + a.y * b.y;
13  }
14
15  i64 cross(Point a, Point b) {
16      return a.x * b.y - a.y * b.x;
17  }
18
19  Point operator+(Point a, Point b) {
20      return Point(a.x + b.x, a.y + b.y);
21  }
22
23  Point operator-(Point a, Point b) {
24      return Point(a.x - b.x, a.y - b.y);
25  }
26
27  auto getHull(std::vector<Point> p) {
28      std::sort(p.begin(), p.end(),
29          [&](auto a, auto b) {
30              return a.x < b.x || (a.x == b.x && a.y < b.y);
31          });
32
33      std::vector<Point> hi, lo;
34      for (auto p : p) {
35          while (hi.size() > 1 && cross(hi.back() - hi[hi.size() - 2], p - hi.back()) >= 0) {
36              hi.pop_back();
37          }
38          while (!hi.empty() && hi.back().x == p.x) {
39              hi.pop_back();
40          }
41          hi.push_back(p);
42          while (lo.size() > 1 && cross(lo.back() - lo[lo.size() - 2], p - lo.back()) <= 0) {
43              lo.pop_back();
44          }
45          if (lo.empty() || lo.back().x < p.x) {
46              lo.push_back(p);
47          }
48      }
49      return std::make_pair(hi, lo);
50  }
51
```

```
const double inf = INFINITY;
```

6.32 11C - 静态凸包（with. std::complex）

Listing 63: **math/11C-Convex-Hull.cpp**

```
/**   静态凸包（with. std::complex）
 *    2022-02-04: https://loj.ac/s/1370861
**/
using Point = std::complex<i64>;

#define x real
#define y imag

auto dot(const Point &a, const Point &b) {
    return (std::conj(a) * b).x();
}

auto cross(const Point &a, const Point &b) {
    return (std::conj(a) * b).y();
}

auto rot(const Point &p) {
    return Point(-p.y(), p.x());
}

auto complexHull(std::vector<Point> a) {
    std::sort(a.begin(), a.end(), [&](auto a, auto b) {
        if (a.x() != b.x()) {
            return a.x() < b.x();
        } else {
            return a.y() < b.y();
        }
    });

    std::vector<Point> l, h;

    for (auto p : a) {
        while (l.size() > 1 && cross(l.back() - l[l.size() - 2], p - l.back()) <= 0) {
            l.pop_back();
        }

        while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0) {
            h.pop_back();
        }

        l.push_back(p);
        h.push_back(p);
    }

    std::reverse(h.begin(), h.end());

    h.insert(h.end(), l.begin() + 1, l.end() - 1);

    return h;
}

int sgn(Point p) {
    if (p.y() > 0 || (p.y() == 0 && p.x() < 0)) {
        return 0;
    } else {
        return 1;
    }
}
```

6.33 12A - 多项式（Poly, with. Z）

Listing 64: **math/12A-Poly.cpp**

```cpp
/**  多项式相关（Poly, with. Z）
 *   2023-02-06: https://atcoder.jp/contests/arc155/submissions/38664055
**/
std::vector<int> rev;
std::vector<Z> roots{0, 1};
void dft(std::vector<Z> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Z u = a[i + j];
                Z v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<Z> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}
struct Poly {
    std::vector<Z> a;
    Poly() {}
    explicit Poly(int size, std::function<Z(int)> f = [](int) { return 0; }) : a(size) {
        for (int i = 0; i < size; i++) {
            a[i] = f(i);
        }
```

```
61      }
62      Poly(const std::vector<Z> &a) : a(a) {}
63      Poly(const std::initializer_list<Z> &a) : a(a) {}
64      int size() const {
65          return a.size();
66      }
67      void resize(int n) {
68          a.resize(n);
69      }
70      Z operator[](int idx) const {
71          if (idx < size()) {
72              return a[idx];
73          } else {
74              return 0;
75          }
76      }
77      Z &operator[](int idx) {
78          return a[idx];
79      }
80      Poly mulxk(int k) const {
81          auto b = a;
82          b.insert(b.begin(), k, 0);
83          return Poly(b);
84      }
85      Poly modxk(int k) const {
86          k = std::min(k, size());
87          return Poly(std::vector<Z>(a.begin(), a.begin() + k));
88      }
89      Poly divxk(int k) const {
90          if (size() <= k) {
91              return Poly();
92          }
93          return Poly(std::vector<Z>(a.begin() + k, a.end()));
94      }
95      friend Poly operator+(const Poly &a, const Poly &b) {
96          std::vector<Z> res(std::max(a.size(), b.size()));
97          for (int i = 0; i < int(res.size()); i++) {
98              res[i] = a[i] + b[i];
99          }
100         return Poly(res);
101     }
102     friend Poly operator-(const Poly &a, const Poly &b) {
103         std::vector<Z> res(std::max(a.size(), b.size()));
104         for (int i = 0; i < int(res.size()); i++) {
105             res[i] = a[i] - b[i];
106         }
107         return Poly(res);
108     }
109     friend Poly operator-(const Poly &a) {
110         std::vector<Z> res(a.size());
111         for (int i = 0; i < int(res.size()); i++) {
112             res[i] = -a[i];
113         }
114         return Poly(res);
115     }
116     friend Poly operator*(Poly a, Poly b) {
117         if (a.size() == 0 || b.size() == 0) {
118             return Poly();
119         }
120         if (a.size() < b.size()) {
121             std::swap(a, b);
122         }
123         if (b.size() < 128) {
```

```
124             Poly c(a.size() + b.size() - 1);
125             for (int i = 0; i < a.size(); i++) {
126                 for (int j = 0; j < b.size(); j++) {
127                     c[i + j] += a[i] * b[j];
128                 }
129             }
130             return c;
131         }
132         int sz = 1, tot = a.size() + b.size() - 1;
133         while (sz < tot) {
134             sz *= 2;
135         }
136         a.a.resize(sz);
137         b.a.resize(sz);
138         dft(a.a);
139         dft(b.a);
140         for (int i = 0; i < sz; ++i) {
141             a.a[i] = a[i] * b[i];
142         }
143         idft(a.a);
144         a.resize(tot);
145         return a;
146     }
147     friend Poly operator*(Z a, Poly b) {
148         for (int i = 0; i < int(b.size()); i++) {
149             b[i] *= a;
150         }
151         return b;
152     }
153     friend Poly operator*(Poly a, Z b) {
154         for (int i = 0; i < int(a.size()); i++) {
155             a[i] *= b;
156         }
157         return a;
158     }
159     Poly &operator+=(Poly b) {
160         return (*this) = (*this) + b;
161     }
162     Poly &operator-=(Poly b) {
163         return (*this) = (*this) - b;
164     }
165     Poly &operator*=(Poly b) {
166         return (*this) = (*this) * b;
167     }
168     Poly &operator*=(Z b) {
169         return (*this) = (*this) * b;
170     }
171     Poly deriv() const {
172         if (a.empty()) {
173             return Poly();
174         }
175         std::vector<Z> res(size() - 1);
176         for (int i = 0; i < size() - 1; ++i) {
177             res[i] = (i + 1) * a[i + 1];
178         }
179         return Poly(res);
180     }
181     Poly integr() const {
182         std::vector<Z> res(size() + 1);
183         for (int i = 0; i < size(); ++i) {
184             res[i + 1] = a[i] / (i + 1);
185         }
186         return Poly(res);
```

```
187     }
188     Poly inv(int m) const {
189         Poly x{a[0].inv()};
190         int k = 1;
191         while (k < m) {
192             k *= 2;
193             x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
194         }
195         return x.modxk(m);
196     }
197     Poly log(int m) const {
198         return (deriv() * inv(m)).integr().modxk(m);
199     }
200     Poly exp(int m) const {
201         Poly x{1};
202         int k = 1;
203         while (k < m) {
204             k *= 2;
205             x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
206         }
207         return x.modxk(m);
208     }
209     Poly pow(int k, int m) const {
210         int i = 0;
211         while (i < size() && a[i].val() == 0) {
212             i++;
213         }
214         if (i == size() || 1LL * i * k >= m) {
215             return Poly(std::vector<Z>(m));
216         }
217         Z v = a[i];
218         auto f = divxk(i) * v.inv();
219         return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
220     }
221     Poly sqrt(int m) const {
222         Poly x{1};
223         int k = 1;
224         while (k < m) {
225             k *= 2;
226             x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
227         }
228         return x.modxk(m);
229     }
230     Poly mulT(Poly b) const {
231         if (b.size() == 0) {
232             return Poly();
233         }
234         int n = b.size();
235         std::reverse(b.a.begin(), b.a.end());
236         return ((*this) * b).divxk(n - 1);
237     }
238     std::vector<Z> eval(std::vector<Z> x) const {
239         if (size() == 0) {
240             return std::vector<Z>(x.size(), 0);
241         }
242         const int n = std::max(int(x.size()), size());
243         std::vector<Poly> q(4 * n);
244         std::vector<Z> ans(x.size());
245         x.resize(n);
246         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
247             if (r - l == 1) {
248                 q[p] = Poly{1, -x[l]};
249             } else {
```

```
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

6.34 12B - 多项式（Poly, with. MInt）

Listing 65: **math/12B-Poly.cpp**

```
/**   多项式相关（Poly, with. MInt & MLong）
 *    2023-09-20: https://atcoder.jp/contests/arc163/submissions/45737810
**/
std::vector<int> rev;
template<int P>
std::vector<MInt<P>> roots{0, 1};

template<int P>
constexpr MInt<P> findPrimitiveRoot() {
    MInt<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}

template<int P>
constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();

template<>
constexpr MInt<998244353> primitiveRoot<998244353> {31};

template<int P>
constexpr void dft(std::vector<MInt<P>> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
```

```
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                MInt<P> u = a[i + j];
                MInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

template<int P>
constexpr void idft(std::vector<MInt<P>> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    MInt<P> inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}

template<int P = 998244353>
struct Poly : public std::vector<MInt<P>> {
    using Value = MInt<P>;

    Poly() : std::vector<Value>() {}
    explicit constexpr Poly(int n) : std::vector<Value>(n) {}

    explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
    constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}

    template<class InputIt, class = std::_RequireInputIter<InputIt>>
    explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(first, last) {}

    template<class F>
    explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
        for (int i = 0; i < n; i++) {
            (*this)[i] = f(i);
        }
    }

    constexpr Poly shift(int k) const {
        if (k >= 0) {
```

```
            auto b = *this;
            b.insert(b.begin(), k, 0);
            return b;
        } else if (this->size() <= -k) {
            return Poly();
        } else {
            return Poly(this->begin() + (-k), this->end());
        }
    }
    constexpr Poly trunc(int k) const {
        Poly f = *this;
        f.resize(k);
        return f;
    }
    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] += b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] -= b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a) {
        std::vector<Value> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
        }
        return Poly(res);
    }
    constexpr friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        if (a.size() < b.size()) {
            std::swap(a, b);
        }
        int n = 1, tot = a.size() + b.size() - 1;
        while (n < tot) {
            n *= 2;
        }
        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
            Poly c(a.size() + b.size() - 1);
            for (int i = 0; i < a.size(); i++) {
                for (int j = 0; j < b.size(); j++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        a.resize(n);
        b.resize(n);
```

```
        dft(a);
        dft(b);
        for (int i = 0; i < n; ++i) {
            a[i] *= b[i];
        }
        idft(a);
        a.resize(tot);
        return a;
    }
    constexpr friend Poly operator*(Value a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    constexpr friend Poly operator*(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    constexpr friend Poly operator/(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] /= b;
        }
        return a;
    }
    constexpr Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    constexpr Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    constexpr Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator*=(Value b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator/=(Value b) {
        return (*this) = (*this) / b;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        Poly res(this->size() - 1);
        for (int i = 0; i < this->size() - 1; ++i) {
            res[i] = (i + 1) * (*this)[i + 1];
        }
        return res;
    }
    constexpr Poly integr() const {
        Poly res(this->size() + 1);
        for (int i = 0; i < this->size(); ++i) {
            res[i + 1] = (*this)[i] / (i + 1);
        }
        return res;
    }
    constexpr Poly inv(int m) const {
        Poly x{(*this)[0].inv()};
        int k = 1;
        while (k < m) {
```

```
            k *= 2;
            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly log(int m) const {
        return (deriv() * inv(m)).integr().trunc(m);
    }
    constexpr Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        Value v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
    }
    constexpr Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
        }
        return x.trunc(m);
    }
    constexpr Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.begin(), b.end());
        return ((*this) * b).shift(-(n - 1));
    }
    constexpr std::vector<Value> eval(std::vector<Value> x) const {
        if (this->size() == 0) {
            return std::vector<Value>(x.size(), 0);
        }
        const int n = std::max(x.size(), this->size());
        std::vector<Poly> q(4 * n);
        std::vector<Value> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
```

```
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template<int P = 998244353>
Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            auto d = oldC;
            d *= -1;
            d.insert(d.begin(), 1);
            MInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}


template<int P = 998244353>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
```

```
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
    }
    return p[0] / q[0];
}

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;

Poly<P> get(int n, int m) {
    if (m == 0) {
        return Poly(n + 1);
    }
    if (m % 2 == 1) {
        auto f = get(n, m - 1);
        Z p = 1;
        for (int i = 0; i <= n; i++) {
            f[n - i] += comb.binom(n, i) * p;
            p *= m;
        }
        return f;
    }
    auto f = get(n, m / 2);
    auto fm = f;
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.fac(i);
    }
    Poly pw(n + 1);
    pw[0] = 1;
    for (int i = 1; i <= n; i++) {
        pw[i] = pw[i - 1] * (m / 2);
    }
    for (int i = 0; i <= n; i++) {
        pw[i] *= comb.invfac(i);
    }
    fm = fm.mulT(pw);
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.invfac(i);
    }
    return f + fm;
}
```

6.35 12C - 多项式乘法

Listing 66: **math/12C-Poly.cpp**

```
/**   多项式乘法
 *    2024-03-10: https://qoj.ac/submission/350298
**/
constexpr int P = 998244353;

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

std::vector<int> rev, roots {0, 1};

void dft(std::vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
```

```
        }
    }
    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (roots.size() < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = 1LL * roots[i] * e % P;
            }
            k++;
        }
    }

    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                int u = a[i + j];
                int v = 1LL * a[i + j + k] * roots[k + j] % P;
                a[i + j] = (u + v) % P;
                a[i + j + k] = (u - v) % P;
            }
        }
    }
}

void idft(std::vector<int> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    int inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * inv % P;
    }
}

std::vector<int> mul(std::vector<int> a, std::vector<int> b) {
    int n = 1, tot = a.size() + b.size() - 1;
    while (n < tot) {
        n *= 2;
    }
    if (tot < 128) {
        std::vector<int> c(a.size() + b.size() - 1);
        for (int i = 0; i < a.size(); i++) {
            for (int j = 0; j < b.size(); j++) {
                c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
            }
        }
        return c;
    }
    a.resize(n);
    b.resize(n);
    dft(a);
    dft(b);
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * b[i] % P;
    }
```

```
88      idft(a);
89      a.resize(tot);
90      return a;
91  }
```

6.36 13A - 生成函数（q-int）

Listing 67: **math/13A-Q-Int.cpp**

```
1   /**  生成函数（q-int）
2    *   2023-09-04: https://qoj.ac/submission/163986
3   **/
4   using i64 = long long;
5   using i128 = __int128;
6
7   i64 power(i64 a, i64 b, i64 p) {
8       i64 res = 1;
9       for (; b; b /= 2, a = i128(a) * a % p) {
10          if (b % 2) {
11              res = i128(res) * a % p;
12          }
13      }
14      return res;
15  }
16
17  std::pair<int, int> qint(int q, int n, int p) {
18      q %= p;
19      for (int x = 2; x * x <= n; x++) {
20          if (n % x == 0) {
21              auto [v1, e1] = qint(q, x, p);
22              auto [v2, e2] = qint(power(q, x, p), n / x, p);
23              return {1LL * v1 * v2 % p, e1 + e2};
24          }
25      }
26      if (q == 1) {
27          if (n == p) {
28              return {0, 1};
29          }
30          return {n, 0};
31      }
32      // std::cerr << q << " " << n << " " << p << "\n";
33      i64 v = 1 - power(q, n, 1LL * p * p);
34      if (v < 0) {
35          v += 1LL * p * p;
36      }
37      assert(v != 0);
38      int inv = power(1 - q + p, p - 2, p);
39      if (v % p == 0) {
40          return {(v / p) * inv % p, 1};
41      } else {
42          return {v % p * inv % p, 0};
43      }
44  }
```

6.37 13B - 生成函数（q-Binomial）

Listing 68: **math/13B-Q-Binomial.cpp**

```
1   /**  生成函数（q-Binomial）
2    *   2023-09-04: https://qoj.ac/submission/164128
3   **/
4   int power(int a, int b, int p) {
5       int res = 1;
```

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

int qint(int n, int q, int p) {
    return 1LL * (power(q, n, p) - 1) * power(q - 1, p - 2, p) % p;
}

int qBinomial(int n, int k, int q, int p) {
    if (q == 0) {
        return 1;
    }
    int r = 0;
    int x = 1;
    do {
        x = 1LL * x * q % p;
        r++;
    } while (x != 1);

    if (n / r > k / r + (n - k) / r) {
        return 0;
    }
    int num = 1, den = 1;
    for (int i = 1; i <= k % r; i++) {
        num = 1LL * num * qint(n % r - i + 1, q, p) % p;
        den = 1LL * den * qint(i, q, p) % p;
    }
    n /= r, k /= r;
    while (n > 0 || k > 0) {
        if (n % p < k % p) {
            return 0;
        }
        for (int i = 1; i <= k % p; i++) {
            num = 1LL * num * (n % p - i + 1) % p;
            den = 1LL * den * i % p;
        }
        n /= p, k /= p;
    }
    int ans = 1LL * num * power(den, p - 2, p) % p;
    return ans;
}
```

6.38 13C - 生成函数（Binomial 任意模数二项式）

Listing 69: **math/13C-Q-Binomial.cpp**

```
/**   生成函数（Binomial 任意模数二项式）
 *    2023-08-22: https://codeforces.com/contest/896/submission/219861532
**/
std::vector<std::pair<int, int>> factorize(int n) {
    std::vector<std::pair<int, int>> factors;
    for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
        if (n % i == 0) {
            int t = 0;
            for (; n % i == 0; n /= i)
                ++t;
            factors.emplace_back(i, t);
        }
    }
    if (n > 1)
```

```
        factors.emplace_back(n, 1);
    return factors;
}
constexpr int power(int base, i64 exp) {
    int res = 1;
    for (; exp > 0; base *= base, exp /= 2) {
        if (exp % 2 == 1) {
            res *= base;
        }
    }
    return res;
}
constexpr int power(int base, i64 exp, int mod) {
    int res = 1 % mod;
    for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
        if (exp % 2 == 1) {
            res = 1LL * res * base % mod;
        }
    }
    return res;
}
int inverse(int a, int m) {
    int g = m, r = a, x = 0, y = 1;
    while (r != 0) {
        int q = g / r;
        g %= r;
        std::swap(g, r);
        x -= q * y;
        std::swap(x, y);
    }
    return x < 0 ? x + m : x;
}
int solveModuloEquations(const std::vector<std::pair<int, int>> &e) {
    int m = 1;
    for (std::size_t i = 0; i < e.size(); i++) {
        m *= e[i].first;
    }
    int res = 0;
    for (std::size_t i = 0; i < e.size(); i++) {
        int p = e[i].first;
        res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
    }
    return res;
}
constexpr int N = 1E5;
class Binomial {
    const int mod;
private:
    const std::vector<std::pair<int, int>> factors;
    std::vector<int> pk;
    std::vector<std::vector<int>> prod;
    static constexpr i64 exponent(i64 n, int p) {
        i64 res = 0;
        for (n /= p; n > 0; n /= p) {
            res += n;
        }
        return res;
    }
    int product(i64 n, std::size_t i) {
        int res = 1;
        int p = factors[i].first;
        for (; n > 0; n /= p) {
            res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) % pk[i] * prod[i][n % pk[i]] % pk[i];
        }
```

```
79          return res;
80      }
81  public:
82      Binomial(int mod) : mod(mod), factors(factorize(mod)) {
83          pk.resize(factors.size());
84          prod.resize(factors.size());
85          for (std::size_t i = 0; i < factors.size(); i++) {
86              int p = factors[i].first;
87              int k = factors[i].second;
88              pk[i] = power(p, k);
89              prod[i].resize(std::min(N + 1, pk[i]));
90              prod[i][0] = 1;
91              for (int j = 1; j < prod[i].size(); j++) {
92                  if (j % p == 0) {
93                      prod[i][j] = prod[i][j - 1];
94                  } else {
95                      prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
96                  }
97              }
98          }
99      }
100     int operator()(i64 n, i64 m) {
101         if (n < m || m < 0) {
102             return 0;
103         }
104         std::vector<std::pair<int, int>> ans(factors.size());
105         for (int i = 0; i < factors.size(); i++) {
106             int p = factors[i].first;
107             int k = factors[i].second;
108             int e = exponent(n, p) - exponent(m, p) - exponent(n - m, p);
109             if (e >= k) {
110                 ans[i] = std::make_pair(pk[i], 0);
111             } else {
112                 int pn = product(n, i);
113                 int pm = product(m, i);
114                 int pd = product(n - m, i);
115                 int res = 1LL * pn * inverse(pm, pk[i]) % pk[i] * inverse(pd, pk[i]) % pk[i] * power(p, e)
                      % pk[i];
116                 ans[i] = std::make_pair(pk[i], res);
117             }
118         }
119         return solveModuloEquations(ans);
120     }
121 };
```

6.39 14 - 自适应辛普森法 Simpson

Listing 70: **math/14-Simpson.cpp**

```
1  /**   自适应辛普森法 Simpson
2   *    2023-09-02: https://qoj.ac/submission/161388
3   **/
4  const double Pi = std::acos(-1.0);
5  constexpr double EPS = 1e-9;
6  double v, r, d;
7  double f(double x) {
8      double s = std::sin(x);
9      return 1 / v / (std::sqrt(s * s + 3) - s);
10 }
11 double simpson(double l, double r) {
12     return (f(l) + 4 * f((l + r) / 2) + f(r)) * (r - l) / 6;
13 }
14 double integral(double l, double r, double eps, double st) {
```

```
    double mid = (l + r) / 2;
    double sl = simpson(l, mid);
    double sr = simpson(mid, r);
    if (std::abs(sl + sr - st) <= 15 * eps)
        return sl + sr + (sl + sr - st) / 15;
    return integral(l, mid, eps / 2, sl) + integral(mid, r, eps / 2, sr);
}
double integral(double l, double r) {
    return integral(l, r, EPS, simpson(l, r));
}
```

6.40 15 - 矩阵（Matrix）

Listing 71: **math/15-Matrix.cpp**

```
/**   矩阵（Matrix）
 *    2024-03-14: https://qoj.ac/submission/353771
**/
using i64 = long long;
using u64 = unsigned long long;

using Matrix = std::array<u64, 65>;

Matrix operator*(const Matrix &a, const Matrix &b) {
    Matrix c{};
    for (int i = 0; i <= 64; i++) {
        for (int j = 0; j <= 64; j++) {
            if (j == 64 ? i == 64 : (a[i] >> j & 1)) {
                c[i] ^= b[j];
            }
        }
    }
    return c;
}

u64 operator*(u64 a, const Matrix &b) {
    u64 c = 0;
    for (int i = 0; i <= 64; i++) {
        if (i == 64 || (a >> i & 1)) {
            c ^= b[i];
        }
    }
    return c;
}

Matrix readMatrix() {
    int m;
    std::cin >> m;

    Matrix f{};
    for (int i = 0; i < m; i++) {
        int s, o;
        u64 A;
        std::cin >> s >> o >> A;

        if (o == 0) {
            for (int j = 0; j < 64; j++) {
                if (A >> ((j + s) % 64) & 1) {
                    f[64] ^= 1ULL << ((j + s) % 64);
                } else {
                    f[j] ^= 1ULL << ((j + s) % 64);
                }
            }
```

```
        } else {
            for (int j = 0; j < 64; j++) {
                if (A >> ((j + s) % 64) & 1) {
                    f[j] ^= 1ULL << ((j + s) % 64);
                }
            }
        }
    }

    u64 B;
    std::cin >> B;
    f[64] ^= B;

    return f;
}
```

6.41 16 - 高斯消元（guess）【久远】

Listing 72: **math/16-Gauss-Elimination.cpp**

```
/**   高斯消元（guess）【久远】
 *    2020-12-02: https://www.codechef.com/viewsolution/39942900
 **/
std::vector<double> gauss(std::vector<std::vector<double>> a, std::vector<double> b) {
    int n = a.size();
    for (int i = 0; i < n; ++i) {
        double x = a[i][i];
        for (int j = i; j < n; ++j) a[i][j] /= x;
        b[i] /= x;
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            x = a[j][i];
            for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
            b[j] -= b[i] * x;
        }
    }
    return b;
}
```

6.42 01A - 树状数组（Fenwick 旧版）

Listing 73: **ds/01A-Fenwick.cpp**

```
/**   树状数组（Fenwick 旧版）
 *    2023-08-11: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=63382128
**/
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

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

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

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

```
        }
    }

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

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

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

6.43 01B - 树状数组（Fenwick 新版）

Listing 74: **ds/01B-Fenwick.cpp**

```
/**   树状数组（Fenwick 新版）
 *    2023-12-28: https://codeforces.com/contest/1915/submission/239262801
**/
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;
    }
};
```

6.44 02 - 并查集（DSU）

Listing 75: **ds/02-DSU.cpp**

```
/**   并查集（DSU）
 *    2023-08-04: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=63239142
**/
struct DSU {
    std::vector<int> f, siz;

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

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

6.45 03A - 线段树（SegmentTree+Info 区间加 + 单点修改）

Listing 76: **ds/03A-Segment-Tree.cpp**

```
/**   线段树（SegmentTree+Info 区间加+单点修改）
 *    2023-09-13: https://qoj.ac/submission/178310
**/
struct SegmentTree {
    int n;
    std::vector<int> tag;
    std::vector<Info> info;
    SegmentTree(int n_) : n(n_), tag(4 * n), info(4 * n) {}

    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }

    void add(int p, int v) {
        tag[p] += v;
        info[p].max += v;
    }

    void push(int p) {
        add(2 * p, tag[p]);
        add(2 * p + 1, tag[p]);
        tag[p] = 0;
    }

    Info query(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return {};
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y);
    }

    Info query(int x, int y) {
        return query(1, 0, n, x, y);
    }

    void rangeAdd(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            return add(p, v);
        }
        int m = (l + r) / 2;
        push(p);
        rangeAdd(2 * p, l, m, x, y, v);
        rangeAdd(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeAdd(int x, int y, int v) {
        rangeAdd(1, 0, n, x, y, v);
    }

    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
```

```
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void modify(int x, const Info &v) {
        modify(1, 0, n, x, v);
    }
};
```

6.46 03B - 线段树（SegmentTree 区间乘 + 单点加）

Listing 77: **ds/03B-Segment-Tree.cpp**

```
/**   线段树（SegmentTree 区间乘+单点加）
 *    2023-10-18: https://cf.dianhsu.com/gym/104417/submission/223800089
**/
struct SegmentTree {
    int n;
    std::vector<int> tag, sum;
    SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) {}

    void pull(int p) {
        sum[p] = (sum[2 * p] + sum[2 * p + 1]) % P;
    }

    void mul(int p, int v) {
        tag[p] = 1LL * tag[p] * v % P;
        sum[p] = 1LL * sum[p] * v % P;
    }

    void push(int p) {
        mul(2 * p, tag[p]);
        mul(2 * p + 1, tag[p]);
        tag[p] = 1;
    }

    int query(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return 0;
        }
        if (l >= x && r <= y) {
            return sum[p];
        }
        int m = (l + r) / 2;
        push(p);
        return (query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y)) % P;
    }

    int query(int x, int y) {
        return query(1, 0, n, x, y);
    }

    void rangeMul(int p, int l, int r, int x, int y, int v) {
        if (l >= y || r <= x) {
```

```
            return;
        }
        if (l >= x && r <= y) {
            return mul(p, v);
        }
        int m = (l + r) / 2;
        push(p);
        rangeMul(2 * p, l, m, x, y, v);
        rangeMul(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeMul(int x, int y, int v) {
        rangeMul(1, 0, n, x, y, v);
    }

    void add(int p, int l, int r, int x, int v) {
        if (r - l == 1) {
            sum[p] = (sum[p] + v) % P;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            add(2 * p, l, m, x, v);
        } else {
            add(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void add(int x, int v) {
        add(1, 0, n, x, v);
    }
};
```

6.47 03C - 线段树（SegmentTree+Info 初始赋值 + 单点修改 + 查找前驱后继）

Listing 78: **ds/03C-Segment-Tree.cpp**

```
/**   线段树（SegmentTree+Info 初始赋值+单点修改+查找前驱后继）
 *    2023-07-17: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=62804432
 *    2024-06-25: https://codeforces.com/contest/1982/submission/267353839
**/
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
```

```
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
```

```
87      int findFirst(int l, int r, F &&pred) {
88          return findFirst(1, 0, n, l, r, pred);
89      }
90      template<class F>
91      int findLast(int p, int l, int r, int x, int y, F &&pred) {
92          if (l >= y || r <= x) {
93              return -1;
94          }
95          if (l >= x && r <= y && !pred(info[p])) {
96              return -1;
97          }
98          if (r - l == 1) {
99              return l;
100         }
101         int m = (l + r) / 2;
102         int res = findLast(2 * p + 1, m, r, x, y, pred);
103         if (res == -1) {
104             res = findLast(2 * p, l, m, x, y, pred);
105         }
106         return res;
107     }
108     template<class F>
109     int findLast(int l, int r, F &&pred) {
110         return findLast(1, 0, n, l, r, pred);
111     }
112 };
```

6.48 03D - 线段树（SegmentTree+Info+Merge 初始赋值 + 单点修改 + 区间合并）

Listing 79: **ds/03D-Segment-Tree.cpp**

```
1  /**   线段树（SegmentTree+Info+Merge 初始赋值+单点修改+区间合并）
2   *    2022-04-23: https://codeforces.com/contest/1672/submission/154766851
3   **/
4  template<class Info,
5      class Merge = std::plus<Info>>
6  struct SegmentTree {
7      const int n;
8      const Merge merge;
9      std::vector<Info> info;
10     SegmentTree(int n) : n(n), merge(Merge()), info(4 << std::__lg(n)) {}
11     SegmentTree(std::vector<Info> init) : SegmentTree(init.size()) {
12         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
13             if (r - l == 1) {
14                 info[p] = init[l];
15                 return;
16             }
17             int m = (l + r) / 2;
18             build(2 * p, l, m);
19             build(2 * p + 1, m, r);
20             pull(p);
21         };
22         build(1, 0, n);
23     }
24     void pull(int p) {
25         info[p] = merge(info[2 * p], info[2 * p + 1]);
26     }
27     void modify(int p, int l, int r, int x, const Info &v) {
28         if (r - l == 1) {
29             info[p] = v;
30             return;
31         }
32         int m = (l + r) / 2;
```

```
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
};
```

6.49 04 - 懒标记线段树（LazySegmentTree）

Listing 80: **ds/04-Lazy-Segt.cpp**

```
/**  懒标记线段树（LazySegmentTree）
 *   2023-03-03: https://atcoder.jp/contests/joi2023yo2/submissions/39363123
 *   2023-03-12: https://codeforces.com/contest/1804/submission/197106837
 *   2023-07-17: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=62804432
 *   2023-11-12: https://qoj.ac/submission/249505
**/
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        tag.assign(4 << std::__lg(n), Tag());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
```

```
35              build(2 * p + 1, m, r);
36              pull(p);
37          };
38          build(1, 0, n);
39      }
40      void pull(int p) {
41          info[p] = info[2 * p] + info[2 * p + 1];
42      }
43      void apply(int p, const Tag &v) {
44          info[p].apply(v);
45          tag[p].apply(v);
46      }
47      void push(int p) {
48          apply(2 * p, tag[p]);
49          apply(2 * p + 1, tag[p]);
50          tag[p] = Tag();
51      }
52      void modify(int p, int l, int r, int x, const Info &v) {
53          if (r - l == 1) {
54              info[p] = v;
55              return;
56          }
57          int m = (l + r) / 2;
58          push(p);
59          if (x < m) {
60              modify(2 * p, l, m, x, v);
61          } else {
62              modify(2 * p + 1, m, r, x, v);
63          }
64          pull(p);
65      }
66      void modify(int p, const Info &v) {
67          modify(1, 0, n, p, v);
68      }
69      Info rangeQuery(int p, int l, int r, int x, int y) {
70          if (l >= y || r <= x) {
71              return Info();
72          }
73          if (l >= x && r <= y) {
74              return info[p];
75          }
76          int m = (l + r) / 2;
77          push(p);
78          return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
79      }
80      Info rangeQuery(int l, int r) {
81          return rangeQuery(1, 0, n, l, r);
82      }
83      void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
84          if (l >= y || r <= x) {
85              return;
86          }
87          if (l >= x && r <= y) {
88              apply(p, v);
89              return;
90          }
91          int m = (l + r) / 2;
92          push(p);
93          rangeApply(2 * p, l, m, x, y, v);
94          rangeApply(2 * p + 1, m, r, x, y, v);
95          pull(p);
96      }
```

```
97      void rangeApply(int l, int r, const Tag &v) {
98          return rangeApply(1, 0, n, l, r, v);
99      }
100     void half(int p, int l, int r) {
101         if (info[p].act == 0) {
102             return;
103         }
104         if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {
105             apply(p, {-(info[p].min + 1) / 2});
106             return;
107         }
108         int m = (l + r) / 2;
109         push(p);
110         half(2 * p, l, m);
111         half(2 * p + 1, m, r);
112         pull(p);
113     }
114     void half() {
115         half(1, 0, n);
116     }
117
118     template<class F>
119     int findFirst(int p, int l, int r, int x, int y, F pred) {
120         if (l >= y || r <= x || !pred(info[p])) {
121             return -1;
122         }
123         if (r - l == 1) {
124             return l;
125         }
126         int m = (l + r) / 2;
127         push(p);
128         int res = findFirst(2 * p, l, m, x, y, pred);
129         if (res == -1) {
130             res = findFirst(2 * p + 1, m, r, x, y, pred);
131         }
132         return res;
133     }
134     template<class F>
135     int findFirst(int l, int r, F pred) {
136         return findFirst(1, 0, n, l, r, pred);
137     }
138     template<class F>
139     int findLast(int p, int l, int r, int x, int y, F pred) {
140         if (l >= y || r <= x || !pred(info[p])) {
141             return -1;
142         }
143         if (r - l == 1) {
144             return l;
145         }
146         int m = (l + r) / 2;
147         push(p);
148         int res = findLast(2 * p + 1, m, r, x, y, pred);
149         if (res == -1) {
150             res = findLast(2 * p, l, m, x, y, pred);
151         }
152         return res;
153     }
154     template<class F>
155     int findLast(int l, int r, F pred) {
156         return findLast(1, 0, n, l, r, pred);
157     }
158
159     void maintainL(int p, int l, int r, int pre) {
```

```
160         if (info[p].difl > 0 && info[p].maxlowl < pre) {
161             return;
162         }
163         if (r - l == 1) {
164             info[p].max = info[p].maxlowl;
165             info[p].maxl = info[p].maxr = l;
166             info[p].maxlowl = info[p].maxlowr = -inf;
167             return;
168         }
169         int m = (l + r) / 2;
170         push(p);
171         maintainL(2 * p, l, m, pre);
172         pre = std::max(pre, info[2 * p].max);
173         maintainL(2 * p + 1, m, r, pre);
174         pull(p);
175     }
176     void maintainL() {
177         maintainL(1, 0, n, -1);
178     }
179     void maintainR(int p, int l, int r, int suf) {
180         if (info[p].difr > 0 && info[p].maxlowr < suf) {
181             return;
182         }
183         if (r - l == 1) {
184             info[p].max = info[p].maxlowl;
185             info[p].maxl = info[p].maxr = l;
186             info[p].maxlowl = info[p].maxlowr = -inf;
187             return;
188         }
189         int m = (l + r) / 2;
190         push(p);
191         maintainR(2 * p + 1, m, r, suf);
192         suf = std::max(suf, info[2 * p + 1].max);
193         maintainR(2 * p, l, m, suf);
194         pull(p);
195     }
196     void maintainR() {
197         maintainR(1, 0, n, -1);
198     }
199 };
200
201 struct Tag {
202     int x = 0;
203     void apply(const Tag &t) & {
204         x = std::max(x, t.x);
205     }
206 };
207
208 struct Info {
209     int x = 0;
210     void apply(const Tag &t) & {
211         x = std::max(x, t.x);
212     }
213 };
214
215 Info operator+(const Info &a, const Info &b) {
216     return {std::max(a.x, b.x)};
217 }
```

6.50 05A - 取模类（Z 旧版）

Listing 81: **ds/05A-ModInt-Old.cpp**

```
constexpr int P = 998244353;
using i64 = long long;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(i64 x) : x(norm(x % P)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
```

```
64          res -= rhs;
65          return res;
66      }
67      friend Z operator/(const Z &lhs, const Z &rhs) {
68          Z res = lhs;
69          res /= rhs;
70          return res;
71      }
72      friend std::istream &operator>>(std::istream &is, Z &a) {
73          i64 v;
74          is >> v;
75          a = Z(v);
76          return is;
77      }
78      friend std::ostream &operator<<(std::ostream &os, const Z &a) {
79          return os << a.val();
80      }
81  };
```

6.51 05B - 取模类（MLong and MInt 新版）

Listing 82: **ds/05B-ModInt-New.cpp**

```
1   template <class T>
2   constexpr T power(T a, i64 b) {
3       T res = 1;
4       for (; b; b /= 2, a *= a) {
5           if (b % 2) {
6               res *= a;
7           }
8       }
9       return res;
10  }
11
12  constexpr i64 mul(i64 a, i64 b, i64 p) {
13      i64 res = a * b - i64(1.L * a * b / p) * p;
14      res %= p;
15      if (res < 0) {
16          res += p;
17      }
18      return res;
19  }
20  template <i64 P>
21  struct MLong {
22      i64 x;
23      constexpr MLong() : x{} {}
24      constexpr MLong(i64 x) : x{norm(x % getMod())} {}
25
26      static i64 Mod;
27      constexpr static i64 getMod() {
28          if (P > 0) {
29              return P;
30          } else {
31              return Mod;
32          }
33      }
34      constexpr static void setMod(i64 Mod_) {
35          Mod = Mod_;
36      }
37      constexpr i64 norm(i64 x) const {
38          if (x < 0) {
39              x += getMod();
40          }
41          if (x >= getMod()) {
```

```
42              x -= getMod();
43          }
44          return x;
45      }
46      constexpr i64 val() const {
47          return x;
48      }
49      explicit constexpr operator i64() const {
50          return x;
51      }
52      constexpr MLong operator-() const {
53          MLong res;
54          res.x = norm(getMod() - x);
55          return res;
56      }
57      constexpr MLong inv() const {
58          assert(x != 0);
59          return power(*this, getMod() - 2);
60      }
61      constexpr MLong &operator*=(MLong rhs) & {
62          x = mul(x, rhs.x, getMod());
63          return *this;
64      }
65      constexpr MLong &operator+=(MLong rhs) & {
66          x = norm(x + rhs.x);
67          return *this;
68      }
69      constexpr MLong &operator-=(MLong rhs) & {
70          x = norm(x - rhs.x);
71          return *this;
72      }
73      constexpr MLong &operator/=(MLong rhs) & {
74          return *this *= rhs.inv();
75      }
76      friend constexpr MLong operator*(MLong lhs, MLong rhs) {
77          MLong res = lhs;
78          res *= rhs;
79          return res;
80      }
81      friend constexpr MLong operator+(MLong lhs, MLong rhs) {
82          MLong res = lhs;
83          res += rhs;
84          return res;
85      }
86      friend constexpr MLong operator-(MLong lhs, MLong rhs) {
87          MLong res = lhs;
88          res -= rhs;
89          return res;
90      }
91      friend constexpr MLong operator/(MLong lhs, MLong rhs) {
92          MLong res = lhs;
93          res /= rhs;
94          return res;
95      }
96      friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
97          i64 v;
98          is >> v;
99          a = MLong(v);
100         return is;
101     }
102     friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) {
103         return os << a.val();
104     }
105     friend constexpr bool operator==(MLong lhs, MLong rhs) {
```

```
106         return lhs.val() == rhs.val();
107     }
108     friend constexpr bool operator!=(MLong lhs, MLong rhs) {
109         return lhs.val() != rhs.val();
110     }
111 };
112
113 template <>
114 i64 MLong<0LL>::Mod = i64(1E18) + 9;
115
116 template <int P>
117 struct MInt {
118     int x;
119     constexpr MInt() : x{} {}
120     constexpr MInt(i64 x) : x{norm(x % getMod())} {}
121
122     static int Mod;
123     constexpr static int getMod() {
124         if (P > 0) {
125             return P;
126         } else {
127             return Mod;
128         }
129     }
130     constexpr static void setMod(int Mod_) {
131         Mod = Mod_;
132     }
133     constexpr int norm(int x) const {
134         if (x < 0) {
135             x += getMod();
136         }
137         if (x >= getMod()) {
138             x -= getMod();
139         }
140         return x;
141     }
142     constexpr int val() const {
143         return x;
144     }
145     explicit constexpr operator int() const {
146         return x;
147     }
148     constexpr MInt operator-() const {
149         MInt res;
150         res.x = norm(getMod() - x);
151         return res;
152     }
153     constexpr MInt inv() const {
154         assert(x != 0);
155         return power(*this, getMod() - 2);
156     }
157     constexpr MInt &operator*=(MInt rhs) & {
158         x = 1LL * x * rhs.x % getMod();
159         return *this;
160     }
161     constexpr MInt &operator+=(MInt rhs) & {
162         x = norm(x + rhs.x);
163         return *this;
164     }
165     constexpr MInt &operator-=(MInt rhs) & {
166         x = norm(x - rhs.x);
167         return *this;
168     }
169     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>;
```

6.52 05C - 动态取模类（ModIntBase）

Listing 83: **ds/05C-Dynamic-ModInt.cpp**

```
/**   动态取模类（ModIntBase）
 *    2024-08-14: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=70980889&returnHomeType=1&uid=329687984
 **/
// TODO: Dynamic ModInt

template <typename T>
constexpr T power(T a, u64 b) {
    T res{1};
    for (; b != 0; b /= 2, a *= a) {
        if (b % 2 == 1) {
            res *= a;
```

```
        }
    }
    return res;
}

template <u32 P>
constexpr u32 mulMod(u32 a, u32 b) {
    return 1ULL * a * b % P;
}

template <u64 P>
constexpr u64 mulMod(u64 a, u64 b) {
    u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
    res %= P;
    return res;
}

template <typename U, U P>
    requires std::unsigned_integral<U>
struct ModIntBase {
public:
    constexpr ModIntBase() : x{0} {}

    template <typename T>
        requires std::integral<T>
    constexpr ModIntBase(T x_) : x{norm(x_ % T{P})} {}

    constexpr static U norm(U x) {
        if ((x >> (8 * sizeof(U) - 1) & 1) == 1) {
            x += P;
        }
        if (x >= P) {
            x -= P;
        }
        return x;
    }

    constexpr U val() const {
        return x;
    }

    constexpr ModIntBase operator-() const {
        ModIntBase res;
        res.x = norm(P - x);
        return res;
    }

    constexpr ModIntBase inv() const {
        return power(*this, P - 2);
    }

    constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
        x = mulMod<P>(x, rhs.val());
        return *this;
    }

    constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }

    constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
        x = norm(x - rhs.x);
        return *this;
```

```
    }

    constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
        return *this *= rhs.inv();
    }

    friend constexpr ModIntBase operator*(ModIntBase lhs, const ModIntBase &rhs) {
        lhs *= rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase &rhs) {
        lhs += rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase &rhs) {
        lhs -= rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase &rhs) {
        lhs /= rhs;
        return lhs;
    }

    friend constexpr std::ostream &operator<<(std::ostream &os, const ModIntBase &a) {
        return os << a.val();
    }

    friend constexpr bool operator==(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() == rhs.val();
    }

    friend constexpr bool operator!=(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() != rhs.val();
    }

    friend constexpr bool operator<(ModIntBase lhs, ModIntBase rhs) {
        return lhs.val() < rhs.val();
    }

private:
    U x;
};

template <u32 P>
using ModInt = ModIntBase<u32, P>;

template <u64 P>
using ModInt64 = ModIntBase<u64, P>;

constexpr u32 P = 998244353;
using Z = ModInt<P>;
```

6.53 06 - 状压 RMQ（RMQ）

Listing 84: **ds/06-RMQ.cpp**

```
/**   状压RMQ（RMQ）
 *    2023-03-02: https://atcoder.jp/contests/joi2022ho/submissions/39351739
 *    2023-09-04: https://qoj.ac/submission/163598
**/
template<class T,
```

```
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
```

```
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

6.54 07A - Splay

Listing 85: **ds/07A-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);
    }
```

```
114 }
115
116 void insert(Tree *&t, Tree *x, Tree *p = nullptr) {
117     if (!t) {
118         t = x;
119         x->p = p;
120         return;
121     }
122
123     push(t);
124     if (x->val < t->val) {
125         insert(t->ch[0], x, t);
126     } else {
127         insert(t->ch[1], x, t);
128     }
129 }
130
131 void dfs(Tree *t) {
132     if (!t) {
133         return;
134     }
135     push(t);
136     dfs(t->ch[0]);
137     std::cerr << t->val << "_";
138     dfs(t->ch[1]);
139 }
140
141 std::pair<Tree *, Tree *> split(Tree *t, int x) {
142     if (!t) {
143         return {t, t};
144     }
145     Tree *v = nullptr;
146     Tree *j = t;
147     for (Tree *i = t; i; ) {
148         push(i);
149         j = i;
150         if (i->val >= x) {
151             v = i;
152             i = i->ch[0];
153         } else {
154             i = i->ch[1];
155         }
156     }
157
158     splay(j);
159     if (!v) {
160         return {j, nullptr};
161     }
162
163     splay(v);
164
165     Tree *u = v->ch[0];
166     if (u) {
167         v->ch[0] = u->p = nullptr;
168     }
169     // std::cerr << "split " << x << "\n";
170     // dfs(u);
171     // std::cerr << "\n";
172     // dfs(v);
173     // std::cerr << "\n";
174     return {u, v};
175 }
176
177 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;
}
```

6.55 07B - Splay

Listing 86: **ds/07B-Splay.cpp**

```
struct Node {
    Node *ch[2], *p;
    bool rev;
    int siz = 1;
    Node() : ch{nullptr, nullptr}, p(nullptr), rev(false) {}
};
void reverse(Node *t) {
    if (t) {
        std::swap(t->ch[0], t->ch[1]);
        t->rev ^= 1;
    }
}
void push(Node *t) {
    if (t->rev) {
        reverse(t->ch[0]);
        reverse(t->ch[1]);
        t->rev = false;
    }
}
void pull(Node *t) {
    t->siz = (t->ch[0] ? t->ch[0]->siz : 0) + 1 + (t->ch[1] ? t->ch[1]->siz : 0);
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void pushAll(Node *t) {
    if (!isroot(t)) {
        pushAll(t->p);
    }
    push(t);
}
void rotate(Node *t) {
    Node *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 (!isroot(q)) {
```

```
        q->p->ch[pos(q)] = t;
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void splay(Node *t) {
    pushAll(t);
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}
void access(Node *t) {
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        i->ch[1] = q;
        pull(i);
    }
    splay(t);
}
void makeroot(Node *t) {
    access(t);
    reverse(t);
}
void link(Node *x, Node *y) {
    makeroot(x);
    x->p = y;
}
void split(Node *x, Node *y) {
    makeroot(x);
    access(y);
}
void cut(Node *x, Node *y) {
    split(x, y);
    x->p = y->ch[0] = nullptr;
    pull(y);
}
int dist(Node *x, Node *y) {
    split(x, y);
    return y->siz - 1;
}
```

6.56 07C - Splay

Listing 87: **ds/07C-Splay.cpp**

```
struct Matrix : std::array<std::array<i64, 4>, 4> {
    Matrix(i64 v = 0) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                (*this)[i][j] = (i == j ? v : inf);
            }
        }
    }
};
```

```
10
11 Matrix operator*(const Matrix &a, const Matrix &b) {
12     Matrix c(inf);
13     for (int i = 0; i < 3; i++) {
14         for (int j = 0; j < 3; j++) {
15             for (int k = 0; k < 4; k++) {
16                 c[i][k] = std::min(c[i][k], a[i][j] + b[j][k]);
17             }
18         }
19         c[i][3] = std::min(c[i][3], a[i][3]);
20     }
21     c[3][3] = 0;
22     return c;
23 }
24
25 struct Node {
26     Node *ch[2], *p;
27     i64 sumg = 0;
28     i64 sumh = 0;
29     i64 sumb = 0;
30     i64 g = 0;
31     i64 h = 0;
32     i64 b = 0;
33     Matrix mat;
34     Matrix prd;
35     std::array<i64, 4> ans{};
36     Node() : ch{nullptr, nullptr}, p(nullptr) {}
37
38     void update() {
39         mat = Matrix(inf);
40         mat[0][0] = b + h - g + sumg;
41         mat[1][1] = mat[1][2] = mat[1][3] = h + sumh;
42         mat[2][0] = mat[2][1] = mat[2][2] = mat[2][3] = b + h + sumb;
43         mat[3][3] = 0;
44     }
45 };
46 void push(Node *t) {
47 }
48 void pull(Node *t) {
49     t->prd = (t->ch[0] ? t->ch[0]->prd : Matrix()) * t->mat * (t->ch[1] ? t->ch[1]->prd : Matrix());
50 }
51 bool isroot(Node *t) {
52     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
53 }
54 int pos(Node *t) {
55     return t->p->ch[1] == t;
56 }
57 void pushAll(Node *t) {
58     if (!isroot(t)) {
59         pushAll(t->p);
60     }
61     push(t);
62 }
63 void rotate(Node *t) {
64     Node *q = t->p;
65     int x = !pos(t);
66     q->ch[!x] = t->ch[x];
67     if (t->ch[x]) {
68         t->ch[x]->p = q;
69     }
70     t->p = q->p;
71     if (!isroot(q)) {
72         q->p->ch[pos(q)] = t;
73     }
```

```
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void splay(Node *t) {
    pushAll(t);
    while (!isroot(t)) {
        if (!isroot(t->p)) {
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}

std::array<i64, 4> get(Node *t) {
    std::array<i64, 4> ans;
    ans.fill(inf);
    ans[3] = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            ans[i] = std::min(ans[i], t->prd[i][j]);
        }
    }
    return ans;
}

void access(Node *t) {
    std::array<i64, 4> old{};
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        if (i->ch[1]) {
            auto res = get(i->ch[1]);
            i->sumg += res[0];
            i->sumh += std::min({res[1], res[2], res[3]});
            i->sumb += std::min({res[0], res[1], res[2], res[3]});
        }
        i->ch[1] = q;
        i->sumg -= old[0];
        i->sumh -= std::min({old[1], old[2], old[3]});
        i->sumb -= std::min({old[0], old[1], old[2], old[3]});
        old = get(i);
        i->update();
        pull(i);
    }
    splay(t);
}
```

```cpp
    /**   Splay
     *    2024-06-24: https://cf.dianhsu.com/gym/105229/submission/267199687?version=1.5
     **/
    constexpr int D = 27;
struct Info {
    int up[D][2]{};
    int down[D][2]{};
    int t = 0;
    i64 ans = 0;
};
```

```

Info operator+(const Info &a, const Info &b) {
    Info c;
    c.t = a.t ^ b.t;
    c.ans = a.ans + b.ans;
    for (int i = 0; i < D; i++) {
        for (int j = 0; j < 2; j++) {
            c.ans += (1LL << i) * a.down[i][j] * b.up[i][j ^ 1];
            c.up[i][j] += a.up[i][j] + b.up[i][j ^ (a.t >> i & 1)];
            c.down[i][j] += b.down[i][j] + a.down[i][j ^ (b.t >> i & 1)];
        }
    }
    return c;
}
struct Node {
    Node *ch[2], *p;
    Info val;
    Info tot;
    int cnt[D][2];
    i64 pair[D][2];
    i64 sum;
    Node() : ch{nullptr, nullptr}, p(nullptr), cnt{}, pair{}, sum{} {}
};
void pull(Node *t) {
    t->tot = (t->ch[0] ? t->ch[0]->tot : Info{}) + t->val + (t->ch[1] ? t->ch[1]->tot : Info{});
}
bool isroot(Node *t) {
    return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
}
int pos(Node *t) {
    return t->p->ch[1] == t;
}
void rotate(Node *t) {
    Node *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 (!isroot(q)) {
        q->p->ch[pos(q)] = t;
    }
    t->ch[x] = q;
    q->p = t;
    pull(q);
}
void update(Node *t) {
    t->val.ans = t->val.t + t->sum;
    for (int i = 0; i < D; i++) {
        t->val.ans += (1LL << i) * t->pair[i][t->val.t >> i & 1];
        for (int j = 0; j < 2; j++) {
            t->val.up[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
            t->val.down[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
        }
        t->val.up[i][t->val.t >> i & 1]++;
        t->val.down[i][t->val.t >> i & 1]++;
    }
    pull(t);
}
void splay(Node *t) {
    while (!isroot(t)) {
        if (!isroot(t->p)) {
```

```
            if (pos(t) == pos(t->p)) {
                rotate(t->p);
            } else {
                rotate(t);
            }
        }
        rotate(t);
    }
    pull(t);
}
void add(Node *t, Info s) {
    for (int i = 0; i < D; i++) {
        for (int x = 0; x < 2; x++) {
            t->pair[i][x] += s.up[i][1 ^ x];
            for (int j = 0; j < 2; j++) {
                t->pair[i][x] += t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
            }
        }
        for (int j = 0; j < 2; j++) {
            t->cnt[i][j] += s.up[i][j];
        }
    }
    t->sum += s.ans;
}
void del(Node *t, Info s) {
    t->sum -= s.ans;
    for (int i = 0; i < D; i++) {
        for (int j = 0; j < 2; j++) {
            t->cnt[i][j] -= s.up[i][j];
        }
        for (int x = 0; x < 2; x++) {
            for (int j = 0; j < 2; j++) {
                t->pair[i][x] -= t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
            }
            t->pair[i][x] -= s.up[i][1 ^ x];
        }
    }
}
void access(Node *t, int v) {
    Info lst;
    for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
        splay(i);
        if (i->ch[1]) {
            add(i, i->ch[1]->tot);
        }
        i->ch[1] = q;
        if (q) {
            del(i, lst);
        } else {
            i->val.t = v;
        }
        lst = i->tot;
        update(i);
    }
    splay(t);
}
```

6.57 08A - 其他平衡树

Listing 88: **ds/08A-Mysterious-Balanced-Tree.cpp**

```
struct Node {
    Node *l = nullptr;
```

```
    Node *r = nullptr;
    int sum = 0;
    int sumodd = 0;

    Node(Node *t) {
        if (t) {
            *this = *t;
        }
    }
};

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

int query1(Node *t1, Node *t2, int l, int r, int k) {
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int odd = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
    int cnt = (t1 && t1->r ? t1->r->sum : 0) - (t2 && t2->r ? t2->r->sum : 0);
    if (odd > 0 || cnt > k) {
        return query1(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
    } else {
        return query1(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
    }
}

std::array<int, 3> query2(Node *t1, Node *t2, int l, int r, int k) {
    if (r - l == 1) {
        int cnt = (t1 ? t1->sumodd : 0) - (t2 ? t2->sumodd : 0);
        return {l, cnt, k};
    }
    int m = (l + r) / 2;
    int cnt = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
    if (cnt > k) {
        return query2(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
    } else {
        return query2(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
    }
}
```

6.58 08B - 其他平衡树

Listing 89: **ds/08B-Mysterious-Balanced-Tree.cpp**

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

```
};

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

int query(Node *t1, Node *t2, int l, int r, int x) {
    int cnt = (t2 ? t2->cnt : 0) - (t1 ? t1->cnt : 0);
    if (cnt == 0 || l >= x) {
        return -1;
    }
    if (r - l == 1) {
        return l;
    }
    int m = (l + r) / 2;
    int res = query(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, x);
    if (res == -1) {
        res = query(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, x);
    }
    return res;
}
```

6.59 08C - 其他平衡树

Listing 90: **ds/08C-Mysterious-Balanced-Tree.cpp**

```
struct Info {
    int imp = 0;
    int id = 0;
};

Info operator+(Info a, Info b) {
    return {std::max(a.imp, b.imp), 0};
}

struct Node {
    int w = rng();
    Info info;
    Info sum;
    int siz = 1;
    Node *l = nullptr;
    Node *r = nullptr;
};

void pull(Node *t) {
    t->sum = t->info;
    t->siz = 1;
    if (t->l) {
        t->sum = t->l->sum + t->sum;
```

```
        t->siz += t->l->siz;
    }
    if (t->r) {
        t->sum = t->sum + t->r->sum;
        t->siz += t->r->siz;
    }
}

std::pair<Node *, Node *> splitAt(Node *t, int p) {
    if (!t) {
        return {t, t};
    }
    if (p <= (t->l ? t->l->siz : 0)) {
        auto [l, r] = splitAt(t->l, p);
        t->l = r;
        pull(t);
        return {l, t};
    } else {
        auto [l, r] = splitAt(t->r, p - 1 - (t->l ? t->l->siz : 0));
        t->r = l;
        pull(t);
        return {t, r};
    }
}

void insertAt(Node *&t, int p, Node *x) {
    if (!t) {
        t = x;
        return;
    }
    if (x->w < t->w) {
        auto [l, r] = splitAt(t, p);
        t = x;
        t->l = l;
        t->r = r;
        pull(t);
        return;
    }
    if (p <= (t->l ? t->l->siz : 0)) {
        insertAt(t->l, p, x);
    } else {
        insertAt(t->r, p - 1 - (t->l ? t->l->siz : 0), x);
    }
    pull(t);
}

Node *merge(Node *a, Node *b) {
    if (!a) {
        return b;
    }
    if (!b) {
        return a;
    }

    if (a->w < b->w) {
        a->r = merge(a->r, b);
        pull(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}
```

```

int query(Node *t, int v) {
    if (!t) {
        return 0;
    }
    if (t->sum.imp < v) {
        return t->siz;
    }
    int res = query(t->r, v);
    if (res != (t->r ? t->r->siz : 0)) {
        return res;
    }
    if (t->info.imp > v) {
        return res;
    }
    return res + 1 + query(t->l, v);
}

void dfs(Node *t) {
    if (!t) {
        return;
    }
    dfs(t->l);
    std::cout << t->info.id << "␣";
    dfs(t->r);
}
```

6.60 08D - 其他平衡树

Listing 91: **ds/08D-Mysterious-Balanced-Tree.cpp**

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

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

struct Info {
    Node *t = nullptr;
    int psum = 0;
    bool rev = false;
};

void pull(Node *t) {
    t->cnt = (t->l ? t->l->cnt : 0) + (t->r ? t->r->cnt : 0);
    t->cntnew = (t->l ? t->l->cntnew : 0) + (t->r ? t->r->cntnew : 0);
}
```

```
34
35 std::pair<Node *, Node *> split(Node *t, int l, int r, int x, bool rev) {
36     if (!t) {
37         return {t, t};
38     }
39     if (x == 0) {
40         return {nullptr, t};
41     }
42     if (x == t->cnt) {
43         return {t, nullptr};
44     }
45     if (r - l == 1) {
46         Node *t2 = new Node;
47         t2->cnt = t->cnt - x;
48         t->cnt = x;
49         return {t, t2};
50     }
51     Node *t2 = new Node;
52     int m = (l + r) / 2;
53     if (!rev) {
54         if (t->l && x <= t->l->cnt) {
55             std::tie(t->l, t2->l) = split(t->l, l, m, x, rev);
56             t2->r = t->r;
57             t->r = nullptr;
58         } else {
59             std::tie(t->r, t2->r) = split(t->r, m, r, x - (t->l ? t->l->cnt : 0), rev);
60         }
61     } else {
62         if (t->r && x <= t->r->cnt) {
63             std::tie(t->r, t2->r) = split(t->r, m, r, x, rev);
64             t2->l = t->l;
65             t->l = nullptr;
66         } else {
67             std::tie(t->l, t2->l) = split(t->l, l, m, x - (t->r ? t->r->cnt : 0), rev);
68         }
69     }
70     pull(t);
71     pull(t2);
72     return {t, t2};
73 }
74
75 Node *merge(Node *t1, Node *t2, int l, int r) {
76     if (!t1) {
77         return t2;
78     }
79     if (!t2) {
80         return t1;
81     }
82     if (r - l == 1) {
83         t1->cnt += t2->cnt;
84         t1->cntnew += t2->cntnew;
85         delete t2;
86         return t1;
87     }
88     int m = (l + r) / 2;
89     t1->l = merge(t1->l, t2->l, l, m);
90     t1->r = merge(t1->r, t2->r, m, r);
91     delete t2;
92     pull(t1);
93     return t1;
94 }
```

6.61 09 - 分数四则运算（Frac）

Listing 92: **ds/09-Frac.cpp**

```
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        if (den < 0) {
            num = -num;
            den = -den;
        }
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.num, a.den);
    }
    friend bool operator==(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator!=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator<(const Frac &lhs, const Frac &rhs) {
```

```
62          return lhs.num * rhs.den < rhs.num * lhs.den;
63      }
64      friend bool operator>(const Frac &lhs, const Frac &rhs) {
65          return lhs.num * rhs.den > rhs.num * lhs.den;
66      }
67      friend bool operator<=(const Frac &lhs, const Frac &rhs) {
68          return lhs.num * rhs.den <= rhs.num * lhs.den;
69      }
70      friend bool operator>=(const Frac &lhs, const Frac &rhs) {
71          return lhs.num * rhs.den >= rhs.num * lhs.den;
72      }
73      friend std::ostream &operator<<(std::ostream &os, Frac x) {
74          T g = std::gcd(x.num, x.den);
75          if (x.den == g) {
76              return os << x.num / g;
77          } else {
78              return os << x.num / g << "/" << x.den / g;
79          }
80      }
81  };
```

6.62 10 - 线性基（Basis）

Listing 93: **ds/10-Basis.cpp**

```
1   struct Basis {
2       int a[20] {};
3       int t[20] {};
4
5       Basis() {
6           std::fill(t, t + 20, -1);
7       }
8
9       void add(int x, int y = 1E9) {
10          for (int i = 0; i < 20; i++) {
11              if (x >> i & 1) {
12                  if (y > t[i]) {
13                      std::swap(a[i], x);
14                      std::swap(t[i], y);
15                  }
16                  x ^= a[i];
17              }
18          }
19      }
20
21      bool query(int x, int y = 0) {
22          for (int i = 0; i < 20; i++) {
23              if ((x >> i & 1) && t[i] >= y) {
24                  x ^= a[i];
25              }
26          }
27          return x == 0;
28      }
29  };
```

6.63 143 - 高精度（BigInt）

Listing 94: **ds/143-BigInt.cpp**

```
1   /**  高精度（BigInt）
2    *   2023-09-11: https://qoj.ac/submission/176420
3    **/
4   constexpr int N = 1000;
```

```

struct BigInt {
    int a[N];
    BigInt(int x = 0) : a{} {
        for (int i = 0; x; i++) {
            a[i] = x % 10;
            x /= 10;
        }
    }
    BigInt &operator*=(int x) {
        for (int i = 0; i < N; i++) {
            a[i] *= x;
        }
        for (int i = 0; i < N - 1; i++) {
            a[i + 1] += a[i] / 10;
            a[i] %= 10;
        }
        return *this;
    }
    BigInt &operator/=(int x) {
        for (int i = N - 1; i >= 0; i--) {
            if (i) {
                a[i - 1] += a[i] % x * 10;
            }
            a[i] /= x;
        }
        return *this;
    }
    BigInt &operator+=(const BigInt &x) {
        for (int i = 0; i < N; i++) {
            a[i] += x.a[i];
            if (a[i] >= 10) {
                a[i + 1] += 1;
                a[i] -= 10;
            }
        }
        return *this;
    }
};

std::ostream &operator<<(std::ostream &o, const BigInt &a) {
    int t = N - 1;
    while (a.a[t] == 0) {
        t--;
    }
    for (int i = t; i >= 0; i--) {
        o << a.a[i];
    }
    return o;
}
```

7 Watashi 代码库 (备用)

7.1 $O(n \log n) - O(1)$ RMQ

Listing 95: **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)];
    }
};
```

7.2 $O(n \log n) - O(\log n)$ LCA

Listing 96: **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;
```

7.3 树状数组

Listing 97: **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));
  }
};
```

7.4 并查集

Listing 98: **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;
    }
};
```

7.5 轻重权树剖分

Listing 99: **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++;
    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;
```

7.6 强连通分量

Listing 100: **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();
```

```
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 };
```

7.7 双连通分量

Listing 101: **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();
```

```
        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);
    }
};
```

7.8 二分图匹配

Listing 102: **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;
    }
};
```

7.9 最小费用最大流

Listing 103: **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];
```

```
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;
```

```
                    Edge *e2 = &e[e1->v][e1->b];
                    e2->c += c;
                    cur = e2->v;
                }
            }
        }
        return make_pair(sumc, sumw);
    }
};
```

7.10 AhoCorasick 自动机

Listing 104: **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();
```

```
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;
```

7.11 后缀数组

Listing 105: **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) {
37                      tmp[j++] = sa[i] - m;
38                  }
39              }
40              // fst
41              fill(cnt.begin(), cnt.end(), 0);
42              for (int i = 0; i < n; ++i) {
43                  ++cnt[rank[i]];
44              }
45              partial_sum(cnt.begin(), cnt.end(), cnt.begin());
46              for (int i = n - 1; i >= 0; --i) {
47                  sa[--cnt[rank[tmp[i]]]] = tmp[i];
48              }
```

```
            //
            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;
            }
        }
    }
};
```

7.12 LU 分解

Listing 106: **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);
```

8 对一类问题的处理方法