



Alfred
代码模版库

目录

1 比赛配置 and 奇技淫巧	4	6.15 07B - 费用流 (MinCostFlow 新版)	43
1.1 多组数据代码模板	4	6.16 08 - 树链剖分 (HLD)	44
1.2 快速读写	4	6.17 01 - 快速幂	46
1.3 关闭流与 C 风格输入输出的同步	4	6.18 02 - 基姆拉尔森公式	47
1.4 .clang-format	5	6.19 03 - 欧拉筛	47
1.5 debug.h	5	6.20 04 - 莫比乌斯函数筛 (莫比乌斯反演)	48
1.6 火车头	7	6.21 05 - 扩展欧几里得 (exgcd)	49
1.7 c-cpp-properties.json	7	6.22 06A - 欧拉函数 (求解单个数的欧拉函数)	49
1.8 launch.json	8	6.23 06B - 欧拉函数 (求解全部数的欧拉函数)	50
1.9 settings.json	8	6.24 07A - 组合数 (小范围预处理, 逆元 + 杨辉三角)	50
1.10 tasks.json	8	6.25 07B - 组合数 (Comb, with. ModInt-Base)	51
2 数据结构	9	6.26 08 - 素数测试与因式分解 (Miller-Rabin and Pollard-Rho)	52
2.1 珂朵莉树	9	6.27 09A - 平面几何 (Point)	53
2.2 树状数组	10	6.28 09B - 平面几何 (with. std::complex)	59
2.3 静态可重区间信息 (支持 RMQ)	11	6.29 10 - 立体几何 (Point)	60
2.4 可删除优先队列	12	6.30 11A - 静态凸包 (with. Point, 旧版)	61
2.5 PBDS 大常数平衡树	13	6.31 11B - 静态凸包 (with. Point, 新版)	62
2.6 离散化容器	13	6.32 11C - 静态凸包 (with. std::complex)	63
2.7 并查集	13	6.33 12A - 多项式 (Poly, with. Z)	64
2.8 可撤销并查集	14	6.34 12B - 多项式 (Poly, with. MInt)	69
2.9 带权并查集	14	6.35 12C - 多项式乘法	76
2.10 出现次数统计	15	6.36 13A - 生成函数 (q-int)	77
2.11 01-Trie	16	6.37 13B - 生成函数 (q-Binomial)	78
2.12 滑动窗口	18	6.38 13C - 生成函数 (Binomial 任意模数二项式)	79
2.13 (二维) 前缀和	18	6.39 14 - 自适应辛普森法 Simpson	81
3 数学 (数论) 算法	20	6.40 15 - 矩阵 (Matrix)	81
3.1 带模整数类	20	6.41 16 - 高斯消元 (guess)【久远】	82
3.2 计算几何	20	6.42 01A - 树状数组 (Fenwick 旧版)	83
3.3 组合数学	22	6.43 01B - 树状数组 (Fenwick 新版)	84
3.4 拉格朗日插值	22	6.44 02 - 并查集 (DSU)	84
3.5 光速幂	23	6.45 03A - 线段树 (SegmentTree+Info 区间加 + 单点修改)	85
4 字符串算法	24	6.46 03B - 线段树 (SegmentTree 区间乘 + 单点加)	86
4.1 字符串哈希	24	6.47 03C - 线段树 (SegmentTree+Info 初始赋值 + 单点修改 + 查找前驱后继)	88
5 图论	24	6.48 03D - 线段树 (SegmentTree+Info+Merge 初始赋值 + 单点修改 + 区间合并)	90
5.1 模块化基本图	24	6.49 04 - 懒标记线段树 (LazySegmentTree)	91
5.2 O(1) LCA	25	6.50 05A - 取模类 (Z 旧版)	94
6 jiangly 代码库 (备用, 侵权请提出 issue)	26	6.51 05B - 取模类 (MLong and MInt 新版)	95
6.1 01 - int128 库函数自定义	26	6.52 05C - 动态取模类 (ModIntBase)	99
6.2 02 - 常用库函数重载	27	6.53 06 - 状压 RMQ (RMQ)	101
6.3 03 - 字符调整	28	6.54 07A - Splay	102
6.4 04A - 二分算法 (整数域)	28	6.55 07B - Splay	105
6.5 04B - 二分算法 (实数域)	29	6.56 07C - Splay	107
6.6 01 - 强连通分量缩点 (SCC)	30	6.57 08A - 其他平衡树	111
6.7 02 - 割边与割边缩点 (EBCC)	31	6.58 08B - 其他平衡树	112
6.8 03 - 二分图最大权匹配 (MaxAssignment 基于 KM)	32	6.59 08C - 其他平衡树	113
6.9 04 - 一般图最大匹配 (Graph 带花树算法)【久远】	34	6.60 08D - 其他平衡树	115
6.10 05 - TwoSat (2-Sat)	36	6.61 09 - 分数四则运算 (Frac)	116
6.11 06A - 最大流 (Flow 旧版其一, 整数应用)	37	6.62 10 - 线性基 (Basis)	117
6.12 06B - 最大流 (Flow 旧版其二, 浮点数应用)	38	6.63 143 - 高精度 (BigInt)	118
6.13 06C - 最大流 (MaxFlow 新版)	40		
6.14 07A - 费用流 (MCFGGraph 旧版)	41		

7	Watashi 代码库 (备用)	119	7.7	双连通分量	124
7.1	$O(n \log n) - O(1)$ RMQ	119	7.8	二分图匹配	126
7.2	$O(n \log n) - O(\log n)$ LCA	120	7.9	最小费用最大流	127
7.3	树状数组	121	7.10	AhoCorasick 自动机	129
7.4	并查集	122	7.11	后缀数组	130
7.5	轻重权树剖分	122	7.12	LU 分解	132
7.6	强连通分量	123	8	对一类问题的处理方法	133

1 比赛配置 and 奇技淫巧

1.1 多组数据代码模板

Listing 1: `template.hpp`

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 using i64 = long long;
4 const i64 N = 1e5 + 10;
5 int t = 1;
6 inline void solve(int Case) {
7     // your code here;
8 }
9 inline void optimizeIO(void) {
10     ios::sync_with_stdio(false);
11     cin.tie(NULL), cout.tie(NULL);
12 }
13 inline void init(void) {}
14 int main(int argc, char const *argv[]) {
15     optimizeIO(), init(), cin >> t;
16     for (int i = 1; i <= t; i++) solve(i);
17     return 0;
18 }

```

1.2 快读快写

Listing 2: `fast-io.hpp`

```

1 #include <bits/stdc++.h>
2
3 namespace fastIO {
4     char c, f, e = 0;
5     namespace usr {
6         template <class _Tp>
7         inline int read(_Tp &x) {
8             x = f = 0, c = getchar();
9             while (!isdigit(c) && !e) f = c == '-', e |= c == EOF, c = getchar();
10            while (isdigit(c) && !e) x = (x << 1) + (x << 3) + (c ^ 48), c = getchar();
11            return (e |= c == EOF) ? 0 : ((f ? x = -x : 0), 1);
12        }
13        template <class _Tp>
14        inline void write(_Tp x) {
15            if (x < 0) putchar('-'), x = -x;
16            if (x > 9) write(x / 10);
17            putchar((x % 10) ^ 48);
18        }
19        template <typename T, typename... V>
20        inline void read(T &t, V &...v) { read(t), read(v...); }
21        template <typename T, typename... V>
22        inline void write(T t, V... v) {
23            write(t), putchar(' '), write(v...);
24        }
25    }
26 }
27 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.hpp`

```

1 #pragma once
2

```

```

3 #include <bits/stdc++.h>
4
5 inline void optimizeIO(void) {
6     std::ios::sync_with_stdio(false);
7     std::cin.tie(NULL), std::cout.tie(NULL);
8 }

```

1.4 .clang-format

Listing 4: .clang-format

```

1 BasedOnStyle: LLVM
2 AlignAfterOpenBracket: BlockIndent
3 # AlignConsecutiveAssignments: Consecutive
4 AlignArrayOfStructures: Right
5 UseTab: Never
6 IndentWidth: 4
7 TabWidth: 4
8 BreakBeforeBraces: Attach
9 AllowShortIfStatementsOnASingleLine: AllIfsAndElse
10 AllowShortLoopsOnASingleLine: true
11 AllowShortBlocksOnASingleLine: true
12 IndentCaseLabels: true
13 ColumnLimit: 0
14 AccessModifierOffset: -4
15 NamespaceIndentation: All
16 FixNamespaceComments: false
17 AllowShortCaseLabelsOnASingleLine: true
18 AlwaysBreakTemplateDeclarations: MultiLine
19 BinPackParameters: true
20 BraceWrapping:
21   AfterCaseLabel: true
22   AfterClass: true
23 AlignConsecutiveMacros: AcrossEmptyLinesAndComments
24 AlignTrailingComments: Always
25 InsertNewlineAtEOF: true

```

1.5 debug.h

Listing 5: debug.h

```

1 /**
2  * @file      debug.h
3  * @author    Dr.Alfred (abonlinejudge@163.com)
4  * @brief     Local Debug Printer
5  * @version   1.0
6  * @date      2023-12-30
7  *
8  * @copyright  Copyright (c) 2019-now <Rhodes Island Inc.>
9  *
10 */
11
12 #include <bits/stdc++.h>
13
14 using std::cerr;
15 using std::pair;
16 using std::string;
17
18 const long long dbg_inf = 9e18 + 19260817;
19
20 void __print(int x) { cerr << x; }
21 void __print(long x) { cerr << x; }
22 void __print(long long x) {
23     if (x != dbg_inf) {

```

```

24     cerr << x;
25 } 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 // To customize a struct/class to print, just define the __print function.
65
66 #if defined(DEBUG) && !defined(NO_DBG_COLOR)
67 #define dbg(x...) \
68     cerr << "\e[91m" << __func__ << ":" << __LINE__ << "_" << #x << "]=_" << " \
69     _print(x); \
70     cerr << "\e[39m"; \
71
72 #define short_dbg(x...) \
73     cerr << "\e[91m[" << " \
74     _print(x); \
75     cerr << "\e[39m"; \
76 #elif defined(DEBUG)
77 #define dbg(x...) \
78     cerr << __func__ << ":" << __LINE__ << "_" << #x << "]=_" << " \
79     _print(x); \
80 #define short_dbg(x...) \
81     cerr << "[" << " \
82     _print(x); \
83 #else
84 #define dbg(x...)
85 #endif // !DEBUG

```

1.6 火车头

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

```

1 #pragma GCC optimize(3)
2 #pragma GCC target("avx")
3 #pragma GCC optimize("Ofast")
4 #pragma GCC optimize("inline")
5 #pragma GCC optimize("-fgcse")
6 #pragma GCC optimize("-fgcse-lm")
7 #pragma GCC optimize("-fipa-sra")
8 #pragma GCC optimize("-ftree-pre")
9 #pragma GCC optimize("-ftree-vrp")
10 #pragma GCC optimize("-fpeephole2")
11 #pragma GCC optimize("-ffast-math")
12 #pragma GCC optimize("-fsched-spec")
13 #pragma GCC optimize("unroll-loops")
14 #pragma GCC optimize("-falign-jumps")
15 #pragma GCC optimize("-falign-loops")
16 #pragma GCC optimize("-falign-labels")
17 #pragma GCC optimize("-fdevirtualize")
18 #pragma GCC optimize("-fcaller-saves")
19 #pragma GCC optimize("-fcrossjumping")
20 #pragma GCC optimize("-fthread-jumps")
21 #pragma GCC optimize("-funroll-loops")
22 #pragma GCC optimize("-fwhole-program")
23 #pragma GCC optimize("-freorder-blocks")
24 #pragma GCC optimize("-fschedule-insns")
25 #pragma GCC optimize("inline-functions")
26 #pragma GCC optimize("-ftree-tail-merge")
27 #pragma GCC optimize("-fschedule-insns2")
28 #pragma GCC optimize("-fstrict-aliasing")
29 #pragma GCC optimize("-fstrict-overflow")
30 #pragma GCC optimize("-falign-functions")
31 #pragma GCC optimize("-fcse-skip-blocks")
32 #pragma GCC optimize("-fcse-follow-jumps")
33 #pragma GCC optimize("-fsched-interblock")
34 #pragma GCC optimize("-fpartial-inlining")
35 #pragma GCC optimize("no-stack-protector")
36 #pragma GCC optimize("-freorder-functions")
37 #pragma GCC optimize("-findirect-inlining")
38 #pragma GCC optimize("-fhoist-adjacent-loads")
39 #pragma GCC optimize("-frerun-cse-after-loop")
40 #pragma GCC optimize("inline-small-functions")
41 #pragma GCC optimize("-finline-small-functions")
42 #pragma GCC optimize("-ftree-switch-conversion")
43 #pragma GCC optimize("-foptimize-sibling-calls")
44 #pragma GCC optimize("-fexpensive-optimizations")
45 #pragma GCC optimize("-funsafe-loop-optimizations")
46 #pragma GCC optimize("inline-functions-called-once")
47 #pragma GCC optimize("-fdelete-null-pointer-checks")

```

1.7 c-cpp-properties.json

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

```

1 {
2   "configurations": [
3     {
4       "name": "macos-gcc-arm64",
5       "includePath": [
6         "${workspaceFolder}/**",

```

```

7         "/usr/local/include/ac-library/"
8     ],
9     "compilerPath": "/usr/local/bin/g++",
10    "cStandard": "c17",
11    "cppStandard": "c++20",
12    "intelliSenseMode": "macos-gcc-arm64",
13    "compilerArgs": [],
14    "configurationProvider": "ms-vscode.makefile-tools"
15 }
16 ],
17 "version": 4
18 }

```

1.8 launch.json

Listing 8: launch.json

```

1 {
2     "version": "0.2.0",
3     "configurations": [
4         {
5             "name": "(lldb)_Launch",
6             "type": "cppdbg",
7             "request": "launch",
8             "program": "${fileDirname}/compiled.out",
9             "args": [],
10            "stopAtEntry": false,
11            "cwd": "dollar{fileDirname}",
12            "environment": [],
13            "externalConsole": true,
14            "internalConsoleOptions": "neverOpen",
15            "MIMode": "lldb",
16            "setupCommands": [
17                {
18                    "description": "Enable pretty-printing for lldb",
19                    "text": "--enable-pretty-printing",
20                    "ignoreFailures": false
21                }
22            ],
23            "preLaunchTask": "Compile"
24        }
25    ],
26 }

```

1.9 settings.json

Listing 9: settings.json

```

1 {
2     "files.defaultLanguage": "cpp",
3     "editor.formatOnType": true,
4     "editor.suggest.snippetsPreventQuickSuggestions": false,
5     "editor.acceptSuggestionOnEnter": "off",
6     "C_Cpp.clang_format_sortIncludes": true,
7     "C_Cpp.errorSquiggles": "disabled",
8     "C_Cpp.default.defines": ["LOCAL", "DEBUG"]
9 }

```

1.10 tasks.json

Listing 10: tasks.json

```

1 // https://code.visualstudio.com/docs/editor/tasks
2 {

```



```

3     "version": "2.0.0",
4     "tasks": [
5         {
6             "label": "Compile", // 任务名称, 与launch.json的preLaunchTask相对应
7             "command": "g++", // 要使用的编译器, C++用g++
8             "args": [
9                 "${file}",
10                "-o", // 指定输出文件名, 不加该参数则默认输出a.exe, Linux下默认a.out
11                "${fileDirname}/compiled.out",
12                "-g", // 生成和调试有关的信息
13                // "-arch aarch64",
14                // "-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
15                "-Wall", // 开启额外警告
16                "-std=c++20", // c++14
17                "-DLOCAL",
18                "-DDEBUG",
19                "-O3",
20                "-ld_classic", // will be deprecated
21                "-Wno-char-subscripts",
22                "-I",
23                "/usr/local/include/ac-library/"
24                // "-stack=268435456" // 手动扩大栈空间
25            ], // 编译的命令, 其实相当于VSC帮你在终端中输了这些东西
26            "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
27                shell再输入命令, 所以args还会经过shell再解析一遍
28            "group": {
29                "kind": "build",
30                "isDefault": true // 不为true时ctrl shift B就要手动选择了
31            },
32            "presentation": {
33                "echo": true,
34                "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
35                    VSC的文档, 即使设为never, 手动点进去还是可以看到
36                "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
37                    有意义
38                "panel": "shared" // 不同的文件的编译信息共享一个终端面板
39            },
40            "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
41                再次触发
42            // 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
43        }
44    ]
45 }

```

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.hpp**

```

1 #pragma once
2
3 #include <bits/stdc++.h>
4
5 struct ChthollyTree {
6     using i64 = long long;
7     struct Node {
8         mutable i64 l, r, v;
9         inline bool operator<(const Node &x) const { return l < x.l; }
10    };
11    std::set<Node> tr;
12    using iterator = std::set<Node>::iterator;
13    ChthollyTree(void) = default;

```

```

14 ChthollyTree(int rng, int val) { init(rng, val); }
15 inline void init(i64 rng, i64 val) noexcept {
16     tr.insert({1, rng, val}), tr.insert({rng + 1, rng + 1, 0});
17 }
18 inline iterator begin(void) const noexcept { return tr.begin(); }
19 inline iterator end(void) const noexcept { return tr.end(); }
20 inline iterator split(i64 pos) {
21     auto it = tr.lower_bound({pos, 0, 0});
22     if (it != tr.end() && it->l == pos) return it;
23     i64 l = (--it)->l, r = it->r, v = it->v;
24     tr.erase(it), tr.insert({l, pos - 1, v});
25     return tr.insert({pos, r, v}).first;
26 }
27 inline void assign(i64 l, i64 r, i64 v) {
28     auto R = split(r + 1), L = split(l);
29     tr.erase(L, R), tr.insert({l, r, v});
30 }
31 template <class _Functor> // func(iterator)
32 inline void modify(i64 l, i64 r, _Functor func) {
33     auto R = split(r + 1), L = split(l);
34     for (auto it = L; it != R; it++) func(it);
35 }
36 template <class _Functor> // func(i64 &, iterator)
37 inline i64 query(i64 l, i64 r, _Functor func) {
38     i64 ans = 0;
39     auto R = split(r + 1);
40     for (auto it = split(l); it != R; it++) func(ans, it);
41     return ans;
42 }
43 };

```

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.hpp

```

1 #pragma once
2
3 #include <bits/stdc++.h>
4
5 template <class T>
6 struct Fenwick {
7     std::vector<T> c;
8     inline int lowbit(int x) { return x & -x; }
9     inline void merge(T &x, T &y) { x = x + y; }
10    inline T subtract(T x, T y) { return x - y; }
11    inline void update(size_t pos, T x) {
12        for (pos++; pos < c.size(); pos += lowbit(pos)) merge(c[pos], x);
13    }
14    inline void clear(void) {
15        for (auto &x : c) x = T();
16    }
17    inline T query(size_t pos) {
18        T ans = T();
19        for (pos++; pos; pos ^= lowbit(pos)) merge(ans, c[pos]);
20        return ans;
21    }
22    inline T query(size_t l, size_t r) {
23        return l == 0 ? query(r) : subtract(query(r), query(l - 1));
24    }
25    inline int kth(const T k) {
26        int ans = 0;

```

```

27     for (int i = 1 << std::__lg(c.size() - 1); i; i >>= 1) {
28         if (ans + i < (int)c.size() && c[ans + i] <= k) {
29             k -= c[ans + i], ans += i;
30         }
31     }
32     return ans;
33 }
34 Fenwick(size_t len) : c(len + 2) {}
35 };

```

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.hpp

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <class T>
6  struct MaxInfo {
7      T val;
8      MaxInfo(void) { val = std::numeric_limits<T>::min(); }
9      template <class InitT>
10     MaxInfo(InitT x) { val = x; }
11     MaxInfo operator+(MaxInfo &x) {
12         return {std::max(val, x.val)};
13     }
14 };
15 template <class T>
16 struct MinInfo {
17     T val;
18     MinInfo(void) { val = std::numeric_limits<T>::max(); }
19     template <class InitT>
20     MinInfo(InitT x) { val = x; }
21     MinInfo operator+(MinInfo &x) {
22         return {std::min(val, x.val)};
23     }
24 };
25 template <class T>
26 struct GcdInfo {
27     T val;
28     GcdInfo(void) { val = T(); }
29     template <class InitT>
30     GcdInfo(InitT x) { val = x; }
31     GcdInfo operator+(GcdInfo &x) {
32 #if __cplusplus >= 201703L
33         return {std::gcd(x.val, val)};
34 #else
35         return {__gcd(x.val, val)};
36 #endif
37     }
38 };
39 template <class T>
40 class SparseTable {
41 private:
42     int n;
43     std::vector<std::vector<T>> ST;
44
45 public:
46     SparseTable(void) {}
47     SparseTable(int N) : n(N), ST(N, std::vector<T>(std::__lg(N) + 1)) {}
48     template <class InitT>

```

```

49 SparseTable(std::vector<InitT> &_init) : SparseTable(_init.size()) { init(_init, true); }
50 template <class InitT>
51 inline void init(std::vector<InitT> &_init, bool internal = false) {
52     if (!internal) {
53         n = _init.size();
54         ST.assign(n, std::vector<T>(std::__lg(n) + 1));
55     }
56     for (int i = 0; i < n; i++) ST[i][0] = T(_init[i]);
57     for (int i = 1; (1 << i) <= n; i++) {
58         for (int j = 0; j + (1 << i) - 1 < n; j++) {
59             ST[j][i] = ST[j][i - 1] + ST[j + (1 << (i - 1))][i - 1];
60         }
61     }
62 }
63 inline T query(int l, int r) { // 0 based
64     if (l > r) return T();
65     int w = std::__lg(r - l + 1);
66     return ST[l][w] + ST[r - (1 << w) + 1][w];
67 }
68 inline T disjoint_query(int l, int r) {
69     T ans = T();
70     for (int i = l; i <= r; i += (1 << std::__lg(r - i + 1))) {
71         ans = ans + ST[i][std::__lg(r - i + 1)];
72     }
73     return ans;
74 }
75 };

```

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.hpp**

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <class T, class Comp = std::less<T>>
6  class PrioritySet { // warning: all erase operations must be legal.
7  private:
8      std::priority_queue<T, std::vector<T>, Comp> data;
9      std::priority_queue<T, std::vector<T>, Comp> erased;
10
11  public:
12      explicit PrioritySet(void) : data(), erased() {};
13      explicit PrioritySet(std::vector<T> &init) {
14          for (auto &v : init) insert(v);
15      }
16      inline void insert(const T &x) { data.push(x); }
17      inline void erase(const T &x) { erased.push(x); }
18      inline T &top(void) noexcept {
19          assert(data.size() >= erased.size());
20          while (!erased.empty() && data.top() == erased.top()) {
21              data.pop(), erased.pop();
22          }
23          return data.top();
24      }
25      inline size_t size(void) {
26          return data.size() - erased.size();
27      }
28  };

```

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.hpp**

```

1  #pragma once
2
3  #include <bits/extc++.h>
4  #include <bits/stdc++.h>
5
6  using namespace std;
7  using namespace __gnu_pbds;
8
9  // TreeTag can also be __gnu_pbds::splay_tree_tag
10 template <class T, class Cmp, class TreeTag = rb_tree_tag>
11 using BalanceTree = tree<T, null_type, Cmp, TreeTag, tree_order_statistics_node_update>;

```

2.6 离散化容器

Listing 16: **discretization.hpp**

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <class _Tp>
6  struct Mess {
7      std::vector<_Tp> v;
8      bool initialized = false;
9      inline _Tp origin(int idx) { return v[idx - 1]; }
10     inline void insert(_Tp x) { v.push_back(x); }
11     template <typename T, typename... V>
12     inline void insert(T x, V... v) { insert(x), insert(v...); }
13     inline void init(void) {
14         sort(v.begin(), v.end()), initialized = true;
15         v.erase(unique(v.begin(), v.end()), v.end());
16     }
17     inline void clear(void) { v.clear(), initialized = false; }
18     inline int query(_Tp x) {
19         if (!initialized) init();
20         return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
21     }
22     inline bool exist(_Tp x) { return origin(query(x)) == x; }
23 };

```

2.7 并查集

Listing 17: **dsu.hpp**

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  struct DSU {
6      std::vector<int> fa, siz;
7      DSU(int n) : fa(n + 1), siz(n + 1, 1) {
8          std::iota(fa.begin(), fa.end(), 0);
9      }
10     inline int find(int x) {
11         return fa[x] == x ? x : fa[x] = find(fa[x]);
12     }
13     inline bool same(int x, int y) {
14         return find(x) == find(y);
15     }
16 };

```

```

15     }
16     // true if x and y were not in the same set, false otherwise.
17     inline bool merge(int x, int y) {
18         int fx = find(x), fy = find(y);
19         if (fx == fy) return false;
20         if (siz[fx] < siz[fy]) std::swap(fx, fy);
21         fa[fy] = fx, siz[fx] += siz[fy], siz[fy] = 0;
22         return true;
23     }
24     // x → y, a.k.a let x be son of y (disable merge by rank).
25     inline bool directed_merge(int x, int y) {
26         int fx = find(x), fy = find(y);
27         if (fx == fy) return false;
28         fa[fx] = fy, siz[fy] += siz[fx], siz[fx] = 0;
29         return true;
30     }
31 };

```

2.8 可撤销并查集

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

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  struct CancelDSU {
6      std::stack<int> S;
7      std::vector<int> fa, siz;
8      CancelDSU(int n) : fa(n + 1), siz(n + 1, 1) {
9          std::iota(fa.begin(), fa.end(), 0);
10     }
11     inline int find(int x) {
12         return fa[x] == x ? x : find(fa[x]);
13     }
14     inline bool same(int x, int y) {
15         return find(x) == find(y);
16     }
17     inline void merge(int u, int v) {
18         int fu = find(u), fv = find(v);
19         if (fu == fv) return S.push(-1);
20         if (siz[fu] < siz[fv]) swap(fu, fv);
21         siz[fu] += siz[fv], fa[fv] = fu, S.push(fv);
22     }
23     inline void _cancel(void) {
24         if (S.empty()) return;
25         if (S.top() == -1) return S.pop();
26         siz[fa[S.top()]] -= siz[S.top()];
27         fa[S.top()] = S.top(), S.pop();
28     }
29     inline void cancel(int t = 1) {
30         while (t--) _cancel();
31     }
32 };

```

2.9 带权并查集

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

```

1  #pragma once
2
3  #include <bits/stdc++.h>

```

```

4
5 template <class T>
6 struct WeightedDSU {
7     std::vector<int> fa;
8     std::vector<T> w;
9     WeightedDSU(int n) : fa(n + 1), w(n + 1) {
10         std::iota(fa.begin(), fa.end(), 0);
11     }
12     inline int find(int x) {
13         if (fa[x] == x) return x;
14         int f = fa[x], f2 = find(f);
15         return w[x] += w[f], fa[x] = f2;
16     }
17     inline bool same(int x, int y) {
18         return find(x) == find(y);
19     }
20     // Given info: a[x] + v = a[y]
21     // Returns true if this operation has no conflict, false otherwise.
22     inline bool merge(int x, int y, T v) {
23         int fx = find(x), fy = find(y);
24         if (fx == fy) {
25             return w[x] + v == w[y];
26         }
27         w[fy] = w[x] + v - w[y], fa[fy] = fx;
28         return true;
29     }
30     inline T distance(int x, int y) {
31         return find(x), find(y), w[y] - w[x];
32     }
33 };

```

2.10 出现次数统计

$O(n \log n)$ 预处理, $O(\log n)$ 查找的出现次数在线统计

Listing 20: `appear-statistics.hpp`

```

1 #pragma once
2
3 #include "discretization.hpp"
4 #include <bits/stdc++.h>
5
6 template <class T>
7 class AppearStats { // Appear Statistics.
8 private:
9     Mess<T> M;
10    size_t n;
11    std::vector<std::vector<int>> pos;
12
13 public:
14     AppearStats(void) : n(0) {}
15     AppearStats(std::vector<T> &init) : n(init.size()) { _init(init); }
16     inline void _init(std::vector<T> &init) {
17         for (auto item : init) M.insert(item);
18         n = init.size(), M.init(), pos.resize(M.v.size());
19         for (size_t i = 0; i < n; i++) {
20             pos[M.query(init[i]) - 1].push_back(i);
21         }
22     }
23     // Use [base] as the beginning of index, return -1 if x doesn't exist.
24     inline int first(int l, int r, T x, int base = 0) {
25         l -= base, r -= base;
26         if (!M.exist(x)) return -1;
27         std::vector<int> &P = pos[M.query(x) - 1];

```

```

28     auto it = std::lower_bound(P.begin(), P.end(), l);
29     return it == P.end() || *it > r ? -1 : *it + base;
30 }
31 // Use [base] as the beginning of index, return -1 if x doesn't exist.
32 inline int last(int l, int r, T x, int base = 0) {
33     l -= base, r -= base;
34     if (!M.exist(x)) return -1;
35     std::vector<int> &P = pos[M.query(x) - 1];
36     auto it = std::upper_bound(P.begin(), P.end(), r);
37     return it == P.begin() || *std::prev(it) < l ? -1 : *std::prev(it) + base;
38 }
39 inline int count(int l, int r, T x, int base = 0) {
40     l -= base, r -= base;
41     if (!M.exist(x)) return 0;
42     std::vector<int> &P = pos[M.query(x) - 1];
43     auto L = std::lower_bound(P.begin(), P.end(), l);
44     auto R = std::upper_bound(P.begin(), P.end(), r);
45     if (L == P.end() || R == P.begin()) return 0;
46     if (*L > r || *std::prev(R) < l) return 0;
47     return R - L;
48 }
49 };

```

2.11 01-Trie

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

```

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

```



```

37     }
38     // Build rootpath of word, insert delta (个) words
39     // Returns BinaryTrie node.
40     int build(uint64_t word, int delta) {
41         int node = ROOT;
42         for (int i = BITS - 1; i >= 0; i--) {
43             tr[node].starting_with += delta;
44             node = get_or_create_child(node, word >> i & 1);
45         }
46         tr[node].starting_with += delta;
47         return node;
48     }
49     // Insert a word with the index of index, INF if index is unknown.
50     // Returns BinaryTrie node.
51     int insert(uint64_t word, int index = INF) {
52         int node = build(word, 1);
53         tr[node].words_here += 1;
54         for (int x = node; x != -1; x = tr[x].parent) {
55             if (index != INF) {
56                 tr[x].min_index = std::min(tr[x].min_index, index);
57                 tr[x].max_index = std::max(tr[x].max_index, index);
58             }
59         }
60         return node;
61     }
62     // Find such an x inserted in the trie that word ^ x is minimized.
63     // Returns such x (x is certain).
64     uint64_t query_min(uint64_t word) {
65         int node = ROOT;
66         uint64_t val = 0;
67         for (int i = BITS - 1; i >= 0; i--) {
68             int go_bit = word >> i & 1;
69             if (tr[node].child[go_bit] == -1) {
70                 go_bit ^= 1;
71             }
72             val |= 1ull << go_bit;
73             node = tr[node].child[go_bit];
74         }
75         return val;
76     }
77     // Find such an x inserted in the trie that word ^ x is maximized.
78     // Returns such x (x is certain).
79     uint64_t query_max(uint64_t word) {
80         int node = ROOT;
81         uint64_t val = 0;
82         for (int i = BITS - 1; i >= 0; i--) {
83             int go_bit = (word >> i & 1) ^ 1;
84             if (tr[node].child[go_bit] == -1) {
85                 go_bit ^= 1;
86             }
87             val |= 1ull << go_bit;
88             node = tr[node].child[go_bit];
89         }
90         return val;
91     }
92     // CF1983F: Find such an x inserted in the trie that word ^ x < upper_bound
93     // Returns a pair {min_index, max_index} of x.
94     std::pair<int, int> query_ub(uint64_t word, uint64_t upper_bound) {
95         int mn = INF, mx = -INF, node = ROOT;
96         for (int i = BITS - 1; i >= 0; i--) {
97             int word_bit = word >> i & 1; // digit i of word
98             int ub_bit = upper_bound >> i & 1; // digit i of ub
99             if (ub_bit == 1 && tr[node].child[word_bit] != -1) {
100                 // if digit i of ub is 1, then we can choose either

```

```

101         // the subtree of word_bit or word_bit ^ 1.
102         mn = std::min(mn, tr[tr[node].child[word_bit]].min_index);
103         mx = std::max(mx, tr[tr[node].child[word_bit]].max_index);
104     }
105     // else if digit i of ub is 0, then we can only choose
106     // the subtree of word_bit. (otherwise, we will violate the range)
107     node = tr[node].child[word_bit ^ ub_bit];
108     if (node == -1) break;
109 }
110 return {mn, mx};
111 }
112 };

```

2.12 滑动窗口

Listing 22: sliding-window.hpp

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <class T> // default max.
6  std::vector<T> sliding_window(std::vector<T> A, size_t k) {
7      std::vector<T> res;
8      std::deque<size_t> Q;
9      for (size_t i = 0; i < A.size(); i++) {
10         if (!Q.empty() && Q[0] + k == i) {
11             Q.pop_front();
12         }
13         while (!Q.empty() && A[Q.back()] <= A[i]) {
14             Q.pop_back();
15         }
16         Q.push_back(i);
17         if (i >= k - 1) { // warning: assert k >= 1
18             res.push_back(A[Q[0]]);
19         }
20     }
21     return res;
22 }
23
24 template <class T>
25 std::vector<std::vector<T>> grid_sliding_window(
26     std::vector<std::vector<T>> &A, size_t x, size_t y
27 ) {
28     const size_t n = A.size(), m = A[0].size();
29     std::vector<std::vector<T>> cols(m - y + 1);
30     std::vector<std::vector<T>> ans(n - x + 1, std::vector<T>(m - y + 1));
31     for (size_t i = 0; i < n; i++) {
32         std::vector<T> res = sliding_window(A[i], y);
33         for (size_t j = 0; j <= m - y; j++) {
34             cols[j].push_back(res[j]);
35         }
36     }
37     for (size_t j = 0; j <= m - y; j++) {
38         std::vector<T> res = sliding_window(cols[j], x);
39         for (size_t i = 0; i <= n - x; i++) {
40             ans[i][j] = res[i];
41         }
42     }
43     return ans;
44 }

```

2.13 (二维) 前缀和

Listing 23: prefix-sum.hpp

```

1  #pragma once
2  #include <bits/stdc++.h>
3
4  template <class T>
5  class Sum {
6  private:
7      size_t n;
8      std::vector<T> sum;
9
10 public:
11     Sum(void) : n(0) {}
12     template <class InitT>
13     Sum(std::vector<InitT> &init) { _init(init); }
14     template <class InitT>
15     inline void _init(std::vector<InitT> &init) {
16         if (init.empty()) return;
17         sum.resize(n = init.size(), sum[0] = init[0]);
18         for (size_t i = 1; i < n; i++) {
19             sum[i] = sum[i - 1] + init[i];
20         }
21     }
22     inline T query(int l, int r) {
23         if (l > r) return T();
24         return l == 0 ? sum[r] : sum[r] - sum[l - 1];
25     }
26 };
27 template <class T>
28 class GridSum {
29 private:
30     size_t n, m;
31     std::vector<std::vector<T>> sum;
32
33 public:
34     GridSum(void) : n(0), m(0) {}
35     template <class InitT>
36     GridSum(std::vector<std::vector<InitT>> &init) { _init(init); }
37     template <class InitT>
38     inline void _init(std::vector<std::vector<InitT>> &init) {
39         if (init.empty()) return;
40         n = init.size(), m = init[0].size();
41         sum.assign(n, std::vector<T>(m)), sum[0][0] = init[0][0];
42         for (size_t i = 1; i < n; i++) {
43             sum[i][0] = sum[i - 1][0] + init[i][0];
44         }
45         for (size_t i = 1; i < m; i++) {
46             sum[0][i] = sum[0][i - 1] + init[0][i];
47         }
48         for (size_t i = 1; i < n; i++) {
49             for (size_t j = 1; j < m; j++) {
50                 sum[i][j] = sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j - 1] + init[i][j];
51             }
52         }
53     }
54     inline T query(int x1, int y1, int x2, int y2) {
55         T s1 = x1 == 0 ? 0 : sum[x1 - 1][y2];
56         T s2 = y1 == 0 ? 0 : sum[x2][y1 - 1];
57         T s3 = x1 == 0 || y1 == 0 ? 0 : sum[x1 - 1][y1 - 1];
58         return sum[x2][y2] - s1 - s2 + s3;
59     }
60 };

```

3 数学（数论）算法

3.1 带模整数类

Listing 24: mod-int.hpp

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <int mod>
6  inline int down(int x) { return x >= mod ? x - mod : x; }
7  template <int mod>
8  struct ModInt {
9      int x;
10     ModInt(void) = default;
11     ModInt(int x) : x(x) {}
12     ModInt(long long x) : x(x % mod) {}
13     friend std::istream &operator>>(std::istream &in, ModInt &a) { return in >> a.x; }
14     friend std::ostream &operator<<(std::ostream &out, ModInt a) { return out << a.x; }
15     friend ModInt operator+(ModInt a, ModInt b) { return down<mod>(a.x + b.x); }
16     friend ModInt operator-(ModInt a, ModInt b) { return down<mod>(a.x - b.x + mod); }
17     friend ModInt operator*(ModInt a, ModInt b) { return (long long)a.x * b.x % mod; }
18     friend ModInt operator/(ModInt a, ModInt b) { return a * ~b; }
19     friend ModInt operator^(ModInt a, long long b) {
20         ModInt ans = 1;
21         for (; b > 0; b >>= 1, a *= a)
22             if (b & 1) ans *= a;
23         return ans;
24     }
25     friend ModInt operator~(ModInt a) { return a ^ (mod - 2); }
26     friend ModInt operator-(ModInt a) { return down<mod>(mod - a.x); }
27     friend ModInt &operator+=(ModInt &a, ModInt b) { return a = a + b; }
28     friend ModInt &operator-=(ModInt &a, ModInt b) { return a = a - b; }
29     friend ModInt &operator*=(ModInt &a, ModInt b) { return a = a * b; }
30     friend ModInt &operator/=(ModInt &a, ModInt b) { return a = a / b; }
31     friend ModInt &operator^=(ModInt &a, long long b) { return a = a ^ b; }
32     friend ModInt &operator++(ModInt &a) { return a += 1; }
33     friend ModInt operator++(ModInt &a, int) {
34         ModInt x = a;
35         a += 1;
36         return x;
37     }
38     friend ModInt &operator--(ModInt &a) { return a -= 1; }
39     friend ModInt operator--(ModInt &a, int) {
40         ModInt x = a;
41         a -= 1;
42         return x;
43     }
44     friend bool operator==(ModInt a, ModInt b) { return a.x == b.x; }
45     friend bool operator!=(ModInt a, ModInt b) { return !(a == b); }
46 };
47 inline void __print(int x) { std::cerr << x; }

```

3.2 计算几何

Listing 25: computation-geometry.hpp

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  // Caution: This computation geometry template is pure shit

```

```

6 //          because of the terrible math level of the author.
7 //          It will be rewritten some time.
8 template <class T>
9 struct Point {
10     T x, y;
11     Point(void) = default;
12     Point(T X, T Y) : x(X), y(Y) {}
13     inline bool operator==(const Point B) {
14         return x == B.x && y == B.y;
15     }
16     friend std::ostream &operator<<(std::ostream &out, Point P) {
17         return out << "(" << P.x << ", " << P.y << ")";
18     }
19     friend std::istream &operator>>(std::istream &in, Point &P) {
20         return in >> P.x >> P.y;
21     }
22 };
23 template <class T>
24 struct Line {
25     T A, B, C; // Ax + By + C = 0
26     Line(void) = default;
27     Line(T a, T b, T c) : A(a), B(b), C(c) {} // Ax + By + C = 0
28     Line(T k, T b) : A(k), B(-1), C(b) {} // y = kx + b
29 };
30 template <class T>
31 inline int sign(T x) {
32     return x == 0 ? 0 : (x < 0 ? -1 : 1);
33 }
34 template <class T>
35 inline bool parallel(Line<T> P, Line<T> Q) {
36     return P.A * Q.B == P.B * Q.A;
37 }
38 template <class T>
39 inline Point<T> intersect(Line<T> P, Line<T> Q) {
40     assert(!parallel(P, Q));
41     return Point<T>{
42         (P.C * Q.B - Q.C * P.B) / (Q.A * P.B - P.A * Q.B),
43         (P.C * Q.A - Q.C * P.A) / (P.A * Q.B - Q.A * P.B)
44     };
45 }
46 template <class T>
47 inline Line<T> get_line(Point<T> P, Point<T> Q) {
48     assert(!(P == Q));
49     if (P.x == Q.x) {
50         return Line<T>(-1, 0, P.x);
51     } else if (P.y == Q.y) {
52         return Line<T>(0, -1, P.y);
53     } else {
54         return Line<T>(
55             Q.y - P.y, P.x - Q.x, P.y * Q.x - P.x * Q.y
56         );
57     }
58 }
59 template <class T>
60 inline bool point_on_line(Point<T> P, Line<T> L) {
61     return L.A * P.x + L.B * P.y + L.C == 0;
62 }
63 template <class T>
64 inline T dis_square(Point<T> P, Point<T> Q) {
65     return (P.x - Q.x) * (P.x - Q.x) + (P.y - Q.y) * (P.y - Q.y);
66 }

```

3.3 组合数学

Listing 26: **comb.hpp**

```

1  #pragma once
2
3  #include "mod-int.hpp"
4  #include <bits/stdc++.h>
5
6  template <class mint>
7  struct Comb {
8      int n;
9      std::vector<mint> _fac, _invfac, _inv;
10     Comb(void) : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
11     Comb(int n) : Comb() { init(n); }
12     inline void init(int m) {
13         _fac.resize(m + 1), _inv.resize(m + 1), _invfac.resize(m + 1);
14         for (int i = n + 1; i <= m; i++) {
15             _fac[i] = _fac[i - 1] * i;
16         }
17         _invfac[m] = ~_fac[m];
18         for (int i = m; i > n; i--) {
19             _invfac[i - 1] = _invfac[i] * i;
20             _inv[i] = _invfac[i] * _fac[i - 1];
21         }
22         n = m;
23     }
24     inline mint fac(int m) {
25         if (m > n) init(m);
26         return _fac[m];
27     }
28     inline mint invfac(int m) {
29         if (m > n) init(m);
30         return _invfac[m];
31     }
32     inline mint inv(int m) {
33         if (m > n) init(m);
34         return _inv[m];
35     }
36     inline mint binom(int n, int m) {
37         if (n < m || m < 0) return 0;
38         return fac(n) * invfac(m) * invfac(n - m);
39     }
40 };
41 Comb<mint> comb;

```

3.4 拉格朗日插值

Listing 27: **lagrange.hpp**

```

1  #pragma once
2
3  #include "comb.hpp"
4  #include "mod-int.hpp"
5  #include <bits/stdc++.h>
6
7  inline mint lagrange(std::vector<mint> &x, std::vector<mint> &y, mint k) {
8      mint ans = 0, cur;
9      const int n = x.size();
10     for (int i = 0; i < n; i++) {
11         cur = y[i];
12         for (int j = 0; j < n; j++) {
13             if (j == i) continue;
14             cur *= (k - x[j]) / (x[i] - x[j]);

```

```

15     }
16     ans += cur;
17 }
18 return ans;
19 }
20 // y[0] is placeholder.
21 // If for all integer x_i in [1, n], we have f(x_i) = y_i (mod p), find f(k) mod p.
22 inline mint cont_lagrange(std::vector<mint> &y, mint k) {
23     mint ans = 0;
24     const int n = y.size() - 1;
25     std::vector<mint> pre(n + 1, 1), suf(n + 2, 1);
26     for (int i = 1; i <= n; i++) pre[i] = pre[i - 1] * (k - i);
27     for (int i = n; i >= 1; i--) suf[i] = suf[i + 1] * (k - i);
28     for (int i = 1; i <= n; i++) {
29         mint A = pre[i - 1] * suf[i + 1];
30         mint B = comb.fac(i - 1) * comb.fac(n - i);
31         ans += ((n - i) & 1 ? -1 : 1) * y[i] * A / B;
32     }
33     return ans;
34 }
35 // find 1^k + 2^k + ... + n^k. in O(k) of time complexity.
36 inline mint sum_of_kth_powers(mint n, int k) {
37     mint sum = 0;
38     std::vector<mint> Y{0};
39     for (int i = 1; i <= k + 2; i++) {
40         Y.push_back(sum += (mint)i ^ k);
41     }
42     return cont_lagrange(Y, n);
43 }

```

3.5 光速幂

Listing 28: speed-of-light-power.hpp

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <int base, int mod>
6  struct SOLPower { // Speed Of Light Power.
7      // p1 stores base^0 ~ base^sq
8      // ps stores base^sq ~ base^(sq^2)
9      std::vector<int> p1, ps;
10     static const int sq = std::sqrt(mod);
11     SOLPower(void) {
12         p1.push_back(1), ps.push_back(1);
13         for (int i = 1; i <= sq; i++) {
14             p1.push_back(1ll * p1.back() * base % mod);
15         }
16         ps.push_back(p1.back());
17         for (int i = 2 * sq; i <= mod; i += sq) {
18             ps.push_back(1ll * ps.back() * ps[1] % mod);
19         }
20     }
21     inline int power(int index) {
22     #if __cplusplus >= 202002L
23         if (index > mod) [[unlikely]] {
24             index %= mod;
25         }
26     #else
27         if (index > mod) index %= mod;
28     #endif
29     return 1ll * ps[index / sq] * p1[index % sq] % mod;

```

```

30     }
31 };

```

4 字符串算法

4.1 字符串哈希

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

```

1  #pragma once
2
3  #include <bits/stdc++.h>
4
5  template <int mod, int seed>
6  struct SingleHash {
7      int n;
8      std::vector<int> pow, h;
9      SingleHash(void) = default;
10     SingleHash(std::string &s) { init(s); }
11     inline void init(std::string &s) {
12         n = s.size(), h.assign(n + 2, 0), pow.assign(n + 2, 1);
13         for (int i = 1; i <= n; i++) {
14             pow[i] = 1ll * pow[i - 1] * seed % mod;
15             h[i] = (1ll * h[i - 1] * seed + s[i - 1]) % mod;
16         }
17     }
18     inline int get_hash(int l, int r) {
19         return (h[r + 1] - 1ll * h[l] * pow[r - l + 1] % mod + mod) % mod;
20     }
21     inline bool check_same(int l1, int r1, int l2, int r2) {
22         return get_hash(l1, r1) == get_hash(l2, r2);
23     }
24 };
25 struct HashedString {
26     SingleHash<998244353, 477> H1;
27     SingleHash<1000000007, 233> H2;
28     HashedString(void) = default;
29     HashedString(std::string s) : H1(s), H2(s) {}
30     inline void init(std::string s) {
31         H1.init(s), H2.init(s);
32     }
33     std::pair<int, int> get_hash(int l, int r) { // not recommended.
34         return {H1.get_hash(l, r), H2.get_hash(l, r)};
35     }
36     // caution: index begins with zero.
37     // If index beginning with one is wanted, use s = ' ' + s
38     inline bool check_same(int l1, int r1, int l2, int r2) {
39         return H1.check_same(l1, r1, l2, r2) && H2.check_same(l1, r1, l2, r2);
40     }
41     inline bool check_period(int l, int r, int p) {
42         return check_same(l, r - p, l + p, r);
43     }
44 };

```

5 图论

5.1 模块化基本图

Listing 30: **graph.hpp**

```

1  #pragma once
2

```



```

3  #include <bits/stdc++.h>
4
5  struct NT {}; // null_type
6  template <class W = NT>
7  class Graph {
8  private:
9      struct Edge {
10         int to;
11         W w;
12     };
13     std::vector<std::vector<Edge>> G; // (to, E)
14
15 public:
16     Graph(void) {}
17     Graph(int n) : G(n + 1) {}
18     inline void clear(void) { G.clear(); }
19     inline void resize(int n) { G.resize(n + 1); }
20     std::vector<Edge> &operator[](int x) { return G[x]; }
21     inline void add_directed(int u, int v, W w = W{}) {
22         G[u].push_back({v, w});
23     }
24     inline void add_undirected(int u, int v, W w = W{}) {
25         G[u].push_back({v, w});
26         G[v].push_back({u, w});
27     }
28     std::vector<W> dijkstra(int s) { // by default the shortest path.
29         using Node = std::pair<W, int>;
30         std::vector<W> dis(G.size(), std::numeric_limits<W>::max());
31         std::priority_queue<Node, std::vector<Node>, std::greater<Node>> heap;
32         heap.push({W{}, s});
33         while (!heap.empty()) {
34             auto t = heap.top();
35             heap.pop();
36             if (dis[t.second] != t.first) {
37                 continue;
38             }
39             for (auto &edge : G[t.second]) {
40                 if (dis[edge.to] > t.first + edge.w) {
41                     dis[edge.to] = t.first + edge.w;
42                     heap.push({dis[edge.to], edge.to});
43                 }
44             }
45         }
46         return dis;
47     }
48 };

```

5.2 $O(1)$ LCA

Listing 31: **lca.hpp**

```

1  #pragma once
2
3  #include "../data_structure/sparse-table.hpp"
4  #include <bits/stdc++.h>
5
6  std::vector<int> G[100010]; // requires a previous graph definition.
7
8  class LCAImpl {
9  private:
10     std::vector<int> dfn, seq; // dfn and seq are (internally) zero indexed.
11     static std::vector<int> d;
12     struct EulerTourInfo {
13         int val;

```

```

14     EulerTourInfo(void) : val(0) {}
15     EulerTourInfo(int x) : val(x) {}
16     EulerTourInfo operator+(EulerTourInfo &x) {
17         return d[val] < d[x.val] ? val : x.val;
18     }
19 };
20 SparseTable<EulerTourInfo> lca; // 0 indexed.
21 inline void _dfs(int x, int fa) {
22     dfn[x] = seq.size();
23     seq.push_back(x), d[x] = d[fa] + 1;
24     for (auto &y : G[x]) {
25         if (y == fa) continue;
26         _dfs(y, x), seq.push_back(x);
27     }
28 }
29
30 public:
31     inline void init(int n) {
32         d.assign(n + 1, 0), dfn.assign(n + 1, 0);
33         seq.clear(), _dfs(1, 0), lca.init(seq);
34     }
35     inline int LCA(int u, int v) {
36         if (u == 0 || v == 0) return u | v;
37         if (dfn[u] > dfn[v]) std::swap(u, v);
38         return lca.query(dfn[u], dfn[v]).val;
39     }
40 } LCA;
41 std::vector<int> LCAImpl::d;

```

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.hpp**

```

1  /**  int128 输出流自定义
2   *   2023-03-20: https://codeforces.com/contest/1806/submission/198413531
3   **/
4  using i128 = __int128;
5
6  std::ostream &operator<<(std::ostream &os, i128 n) {
7      std::string s;
8      while (n) {
9          s += '0' + n % 10;
10         n /= 10;
11     }
12     std::reverse(s.begin(), s.end());
13     return os << s;
14 }
15
16 std::istream &operator>>(std::istream &is, i128 &n) {
17     std::string s;
18     is >> s;
19     for (auto &c : s) {
20         n = n * 10 + c - '0';
21     }
22     return is;
23 }
24
25 i128 toi128(const std::string &s) {
26     i128 n = 0;
27     for (auto c : s) {
28         n = n * 10 + (c - '0');

```

```

29     }
30     return n;
31 }
32
33 i128 sqrti128(i128 n) {
34     i128 lo = 0, hi = 1E16;
35     while (lo < hi) {
36         i128 x = (lo + hi + 1) / 2;
37         if (x * x <= n) {
38             lo = x;
39         } else {
40             hi = x - 1;
41         }
42     }
43     return lo;
44 }

```

6.2 02 - 常用库函数重载

Listing 33: `others/02-Math-Func.hpp`

```

1  using i64 = long long;
2  using i128 = __int128;
3
4  /** 上取整下取整
5   * 2023-10-15: https://codeforces.com/contest/293/submission/228297248
6   **/
7  i64 ceilDiv(i64 n, i64 m) {
8      if (n >= 0) {
9          return (n + m - 1) / m;
10     } else {
11         return n / m;
12     }
13 }
14
15 i64 floorDiv(i64 n, i64 m) {
16     if (n >= 0) {
17         return n / m;
18     } else {
19         return (n - m + 1) / m;
20     }
21 }
22
23 /** 最大值赋值
24 * 2023-09-30: https://codeforces.com/contest/1874/submission/226069129
25 **/
26 template<class T>
27 void chmax(T &a, T b) {
28     if (a < b) {
29         a = b;
30     }
31 }
32
33 /** 最大公约数
34 * -: -
35 **/
36 i128 gcd(i128 a, i128 b) {
37     return b ? gcd(b, a % b) : a;
38 }
39
40 /** 精确开平方
41 * 2024-03-02: https://qoj.ac/submission/343317
42 **/
43 i64 sqrt(i64 n) {

```

```

44     i64 s = std::sqrt(n);
45     while (s * s > n) {
46         s--;
47     }
48     while ((s + 1) * (s + 1) <= n) {
49         s++;
50     }
51     return s;
52 }
53
54 /** 精确开平方
55  * 2023-09-19: https://qoj.ac/submission/183430
56 **/
57 i64 get(i64 n) {
58     i64 u = std::sqrt(2.0L * n);
59     while (u * (u + 1) / 2 < n) {
60         u++;
61     }
62     while (u * (u - 1) / 2 + 1 > n) {
63         u--;
64     }
65     return u;
66 }

```

6.3 03 - 字符调整

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

```

1  /** 大小写转换、获取字母序
2  * 2024-03-16: https://qoj.ac/submission/355156
3  **/
4  void rev(std::string &s) {
5      int l = s.size();
6      for (int i = 1; i < l; i += 2) {
7          if (std::isupper(s[i])) {
8              s[i] = std::tolower(s[i]);
9          } else {
10             s[i] = std::toupper(s[i]);
11         }
12     }
13 }
14
15 int get(char c) {
16     int x;
17     if (std::islower(c)) {
18         x = c - 'a';
19     } else {
20         x = 26 + c - 'A';
21     }
22     return x;
23 }

```

6.4 04A - 二分算法 (整数域)

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

```

1  /** 二分算法 (整数域): 前驱
2  * 2023-09-18: https://qoj.ac/submission/182628
3  **/
4  int lo = 1, hi = 1E9;
5  while (lo < hi) {
6      int m = (lo + hi + 1) / 2;
7      if (check(m)) {

```

```

8         lo = m;
9     } else {
10         hi = m - 1;
11     }
12 }
13 std::cout << lo << "\n";
14
15 /** 二分算法 (整数域): 后继
16 * 2023-09-18: https://qoj.ac/submission/182752
17 **/
18 int lo = 1, hi = n;
19 while (lo < hi) {
20     int m = (lo + hi) / 2;
21     if (check(m)) {
22         hi = m;
23     } else {
24         lo = m + 1;
25     }
26 }
27 std::cout << lo << "\n";

```

6.5 04B - 二分算法 (实数域)

Listing 36: `others/04B-Binary-Search.hpp`

```

1 /** 二分算法 (实数域)
2 * 2023-10-21: https://qoj.ac/submission/222042
3 **/
4 auto check = [&](double t) {
5     // write
6 };
7
8 double lo = 0;
9 double hi = 1E12;
10 while (hi - lo > std::max(1.0, lo) * eps) {
11     double x = (lo + hi) / 2;
12     if (check(x)) {
13         hi = x;
14     } else {
15         lo = x;
16     }
17 }
18
19 std::cout << lo << "\n";
20
21 /** 二分算法 (实数域)
22 * 2023-09-15: https://qoj.ac/submission/179994
23 **/
24 using i64 = long long;
25 using real = long double;
26
27 constexpr real eps = 1E-7;
28
29 auto get = [&](const auto &f) {
30     real lo = -1E4, hi = 1E4;
31     while (hi - lo > 3 * eps) {
32         real x1 = (lo + hi - eps) / 2;
33         real x2 = (lo + hi + eps) / 2;
34         if (f(x1) > f(x2)) {
35             lo = x1;
36         } else {
37             hi = x2;
38         }
39     }

```

```

40     return f((lo + hi) / 2);
41 };
42
43 std::cout << get([&](real px) {
44     return get([&](real py) {
45         // write
46     });
47 }) << "\n";

```

6.6 01 - 强连通分量缩点 (SCC)

Listing 37: graph/01-SCC.hpp

```

1  /** 强连通分量缩点 (SCC)
2   *   2023-06-18: https://codeforces.com/contest/1835/submission/210147209
3   */
4  struct SCC {
5      int n;
6      std::vector<std::vector<int>> adj;
7      std::vector<int> stk;
8      std::vector<int> dfn, low, bel;
9      int cur, cnt;
10
11      SCC() {}
12      SCC(int n) {
13          init(n);
14      }
15
16      void init(int n) {
17          this->n = n;
18          adj.assign(n, {});
19          dfn.assign(n, -1);
20          low.resize(n);
21          bel.assign(n, -1);
22          stk.clear();
23          cur = cnt = 0;
24      }
25
26      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.hpp

```

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);
23         bel.assign(n, -1);
24         stk.clear();
25         cur = cnt = 0;
26     }
27
28     void addEdge(int u, int v) {
29         adj[u].push_back(v);
30         adj[v].push_back(u);
31     }
32
33     void dfs(int x, int p) {
34         dfn[x] = low[x] = cur++;
35         stk.push_back(x);
36
37         for (auto y : adj[x]) {
38             if (y == p) {
39                 continue;
40             }
41             if (dfn[y] == -1) {
42                 E.emplace(x, y);
43                 dfs(y, x);
44                 low[x] = std::min(low[x], low[y]);
45             } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
46                 E.emplace(x, y);

```

```

47         low[x] = std::min(low[x], dfn[y]);
48     }
49 }
50
51 if (dfn[x] == low[x]) {
52     int y;
53     do {
54         y = stk.back();
55         bel[y] = cnt;
56         stk.pop_back();
57     } while (y != x);
58     cnt++;
59 }
60 }
61
62 std::vector<int> work() {
63     dfs(0, -1);
64     return bel;
65 }
66
67 struct Graph {
68     int n;
69     std::vector<std::pair<int, int>> edges;
70     std::vector<int> siz;
71     std::vector<int> cnte;
72 };
73 Graph compress() {
74     Graph g;
75     g.n = cnt;
76     g.siz.resize(cnt);
77     g.cnte.resize(cnt);
78     for (int i = 0; i < n; i++) {
79         g.siz[bel[i]]++;
80         for (auto j : adj[i]) {
81             if (bel[i] < bel[j]) {
82                 g.edges.emplace_back(bel[i], bel[j]);
83             } else if (i < j) {
84                 g.cnte[bel[i]]++;
85             }
86         }
87     }
88     return g;
89 }
90 };

```

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

Listing 39: graph/03-Max-Assignment.hpp

```

1  /** 二分图最大权匹配 (MaxAssignment 基于KM)
2   *   2022-04-10: https://atcoder.jp/contests/abc247/submissions/30867023
3   *   2023-09-21: https://qoj.ac/submission/184824
4   **/
5  constexpr int inf = 1E7;
6  template<class T>
7  struct MaxAssignment {
8      public:
9          T solve(int nx, int ny, std::vector<std::vector<T>> a) {
10              assert(0 <= nx && nx <= ny);
11              assert(int(a.size()) == nx);
12              for (int i = 0; i < nx; ++i) {
13                  assert(int(a[i].size()) == ny);
14                  for (auto x : a[i])
15                      assert(x >= 0);

```



```

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

```

```

79         lx[x] -= delta;
80     for (int y = 0; y < ny; ++y) {
81         if (visy[y]) {
82             ly[y] += delta;
83         } else {
84             slack[y] -= delta;
85         }
86     }
87     for (int y = 0; y < ny; ++y) {
88         if (!visy[y] && slack[y] == 0) {
89             if (yx[y] == -1) {
90                 ex = slackx[y];
91                 ey = y;
92                 found = true;
93                 break;
94             }
95             que.push(yx[y]);
96             p[yx[y]] = slackx[y];
97             visy[y] = visx[yx[y]] = true;
98             update(yx[y]);
99         }
100     }
101 }
102
103 costs[cur + 1] = costs[cur];
104 for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
105     costs[cur + 1] += a[x][y];
106     if (xy[x] != -1)
107         costs[cur + 1] -= a[x][xy[x]];
108     ty = xy[x];
109     xy[x] = y;
110     yx[y] = x;
111 }
112 }
113 return costs[nx];
114 }
115 std::vector<int> assignment() {
116     return xy;
117 }
118 std::pair<std::vector<T>, std::vector<T>> labels() {
119     return std::make_pair(lx, ly);
120 }
121 std::vector<T> weights() {
122     return costs;
123 }
124 private:
125     std::vector<T> lx, ly, slack, costs;
126     std::vector<int> xy, yx, p, slackx;
127     std::vector<bool> visx, visy;
128 };

```

6.9 04 - 一般图最大匹配 (Graph 带花树算法)【久远】

Listing 40: graph/04-Graph-Match.hpp

```

1  /** 一般图最大匹配 (Graph 带花树算法) 【久远】
2  *   2021-12-24: https://codeforces.com/contest/1615/submission/140509278
3  **/
4  struct Graph {
5      int n;
6      std::vector<std::vector<int>> e;
7      Graph(int n) : n(n), e(n) {}
8      void addEdge(int u, int v) {
9          e[u].push_back(v);

```

```

10     e[v].push_back(u);
11 }
12 std::vector<int> findMatching() {
13     std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
14
15     // disjoint set union
16     auto find = [&](int u) {
17         while (f[u] != u)
18             u = f[u] = f[f[u]];
19         return u;
20     };
21
22     auto lca = [&](int u, int v) {
23         u = find(u);
24         v = find(v);
25         while (u != v) {
26             if (dep[u] < dep[v])
27                 std::swap(u, v);
28             u = find(link[match[u]]);
29         }
30         return u;
31     };
32
33     std::queue<int> que;
34     auto blossom = [&](int u, int v, int p) {
35         while (find(u) != p) {
36             link[u] = v;
37             v = match[u];
38             if (vis[v] == 0) {
39                 vis[v] = 1;
40                 que.push(v);
41             }
42             f[u] = f[v] = p;
43             u = link[v];
44         }
45     };
46
47     // find an augmenting path starting from u and augment (if exist)
48     auto augment = [&](int u) {
49
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.hpp`

```

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.hpp

```

1  /** 最大流 (Flow 旧版其一, 整数应用)
2   *   2022-09-03: https://codeforces.com/contest/1717/submission/170688062
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     for (int &i = cur[u]; i < int(g[u].size()); ++i) {
45         const int j = g[u][i];
46         auto [v, c] = e[j];
47         if (c > 0 && h[v] == h[u] + 1) {
48             auto a = dfs(v, t, std::min(r, c));
49             e[j].cap -= a;
50             e[j ^ 1].cap += a;
51             r -= a;
52             if (r == 0) {
53                 return f;
54             }
55         }
56     }
57     return f - r;
58 }
59 void addEdge(int u, int v, T c) {
60     g[u].push_back(e.size());
61     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.hpp

```

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.hpp

```

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.hpp

```

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 MCFGGraph {
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<
20             i64, int>>> que;
21         dis[s] = 0;
22         que.emplace(0, s);
23         while (!que.empty()) {
24             i64 d = que.top().first;
25             int u = que.top().second;
26             que.pop();
27             if (dis[u] < d) continue;
28             for (int i : g[u]) {
29                 int v = e[i].v;
30                 int c = e[i].c;
31                 int f = e[i].f;
32                 if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
33                     dis[v] = d + h[u] - h[v] + f;
34                     pre[v] = i;
35                     que.emplace(dis[v], v);
36                 }
37             }
38         }
39         return dis[t] != std::numeric_limits<i64>::max();
40     }
41     MCFGGraph(int n) : n(n), g(n) {}
42     void addEdge(int u, int v, int c, int f) {
43         // if (f < 0) {
44             g[u].push_back(e.size());
45             e.emplace_back(v, 0, f);
46             g[v].push_back(e.size());
47             e.emplace_back(u, c, -f);
48         // } else {
49             // g[u].push_back(e.size());
50             // e.emplace_back(v, c, f);
51             // g[v].push_back(e.size());
52             // e.emplace_back(u, 0, -f);
53         // }
54     }
55     std::pair<int, i64> flow(int s, int t) {
56         int flow = 0;
57         i64 cost = 0;
58         h.assign(n, 0);
59         while (dijkstra(s, t)) {
60             for (int i = 0; i < n; ++i) h[i] += dis[i];
61             int aug = std::numeric_limits<int>::max();
62             for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug, e[pre[i]].c);
63             for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
64                 e[pre[i]].c -= aug;
65                 e[pre[i] ^ 1].c += aug;

```

```

65         }
66         flow += aug;
67         cost += i64(aug) * h[t];
68     }
69     return std::make_pair(flow, cost);
70 }
71 };

```

6.15 07B - 费用流 (MinCostFlow 新版)

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

```

1  /** MinCostFlow 新版
2  *   2023-11-09: https://qoj.ac/submission/244680
3  **/
4  template<class T>
5  struct MinCostFlow {
6      struct _Edge {
7          int to;
8          T cap;
9          T cost;
10         _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
11     };
12     int n;
13     std::vector<_Edge> e;
14     std::vector<std::vector<int>> g;
15     std::vector<T> h, dis;
16     std::vector<int> pre;
17     bool dijkstra(int s, int t) {
18         dis.assign(n, std::numeric_limits<T>::max());
19         pre.assign(n, -1);
20         std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::greater<std::pair<T,
21             int>>> que;
22         dis[s] = 0;
23         que.emplace(0, s);
24         while (!que.empty()) {
25             T d = que.top().first;
26             int u = que.top().second;
27             que.pop();
28             if (dis[u] != d) {
29                 continue;
30             }
31             for (int i : g[u]) {
32                 int v = e[i].to;
33                 T cap = e[i].cap;
34                 T cost = e[i].cost;
35                 if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
36                     dis[v] = d + h[u] - h[v] + cost;
37                     pre[v] = i;
38                     que.emplace(dis[v], v);
39                 }
40             }
41         }
42         return dis[t] != std::numeric_limits<T>::max();
43     }
44     MinCostFlow() {}
45     MinCostFlow(int n_) {
46         init(n_);
47     }
48     void init(int n_) {
49         n = n_;
50         e.clear();
51         g.assign(n, {});

```

```

52 void addEdge(int u, int v, T cap, T cost) {
53     g[u].push_back(e.size());
54     e.emplace_back(v, cap, cost);
55     g[v].push_back(e.size());
56     e.emplace_back(u, 0, -cost);
57 }
58 std::pair<T, T> flow(int s, int t) {
59     T flow = 0;
60     T cost = 0;
61     h.assign(n, 0);
62     while (dijkstra(s, t)) {
63         for (int i = 0; i < n; ++i) {
64             h[i] += dis[i];
65         }
66         T aug = std::numeric_limits<int>::max();
67         for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
68             aug = std::min(aug, e[pre[i]].cap);
69         }
70         for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
71             e[pre[i]].cap -= aug;
72             e[pre[i] ^ 1].cap += aug;
73         }
74         flow += aug;
75         cost += aug * h[t];
76     }
77     return std::make_pair(flow, cost);
78 }
79 struct Edge {
80     int from;
81     int to;
82     T cap;
83     T cost;
84     T flow;
85 };
86 std::vector<Edge> edges() {
87     std::vector<Edge> a;
88     for (int i = 0; i < e.size(); i += 2) {
89         Edge x;
90         x.from = e[i + 1].to;
91         x.to = e[i].to;
92         x.cap = e[i].cap + e[i + 1].cap;
93         x.cost = e[i].cost;
94         x.flow = e[i + 1].cap;
95         a.push_back(x);
96     }
97     return a;
98 }
99 };

```

6.16 08 - 树链剖分 (HLD)

Listing 47: graph/08-HLD.hpp

```

1  /** 树链剖分 (HLD)
2   *   2023-08-31: https://codeforces.com/contest/1863/submission/221214363
3   **/
4  struct HLD {
5      int n;
6      std::vector<int> siz, top, dep, parent, in, out, seq;
7      std::vector<std::vector<int>> adj;
8      int cur;
9
10     HLD() {}
11     HLD(int n) {

```

```

12     init(n);
13 }
14 void init(int n) {
15     this->n = n;
16     siz.resize(n);
17     top.resize(n);
18     dep.resize(n);
19     parent.resize(n);
20     in.resize(n);
21     out.resize(n);
22     seq.resize(n);
23     cur = 0;
24     adj.assign(n, {});
25 }
26 void addEdge(int u, int v) {
27     adj[u].push_back(v);
28     adj[v].push_back(u);
29 }
30 void work(int root = 0) {
31     top[root] = root;
32     dep[root] = 0;
33     parent[root] = -1;
34     dfs1(root);
35     dfs2(root);
36 }
37 void dfs1(int u) {
38     if (parent[u] != -1) {
39         adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
40     }
41
42     siz[u] = 1;
43     for (auto &v : adj[u]) {
44         parent[v] = u;
45         dep[v] = dep[u] + 1;
46         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 isAncestor(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 (!isAncestor(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) {
111            return n;
112        }
113        if (!isAncestor(v, u)) {
114            return siz[v];
115        }
116        return n - siz[rootedParent(u, v)];
117    }
118
119    int rootedLca(int a, int b, int c) {
120        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
121    }
122 };

```

6.17 01 - 快速幂

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

```

1  /** 快速幂 - 普通版
2   *   2023-10-09: https://atcoder.jp/contests/tenka1-2017/submissions/46411797
3   **/
4  int power(int a, i64 b, int p) {
5      int res = 1;
6      for (; b /= 2, a = 1LL * a * a % p) {
7          if (b % 2) {
8              res = 1LL * res * a % p;
9          }
10     }
11     return res;

```

```

12 }
13
14 /** 快速幂 - 手写乘法
15 * 2023-09-27: https://qoj.ac/submission/189343
16 **/
17 using i64 = long long;
18
19 i64 mul(i64 a, i64 b, i64 p) {
20     i64 c = a * b - i64(1.0L * a * b / p) * p;
21     c %= p;
22     if (c < 0) {
23         c += p;
24     }
25     return c;
26 }
27
28 i64 power(i64 a, i64 b, i64 p) {
29     i64 res = 1;
30     for (; b; b /= 2, a = mul(a, a, p)) {
31         if (b % 2) {
32             res = mul(res, a, p);
33         }
34     }
35     return res;
36 }

```

6.18 02 - 基姆拉尔森公式

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

```

1 /** 基姆拉尔森公式
2 * 2023-09-05: https://qoj.ac/submission/164735
3 **/
4 const int d[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
5
6 bool isLeap(int y) {
7     return y % 400 == 0 || (y % 4 == 0 && y % 100 != 0);
8 }
9
10 int daysInMonth(int y, int m) {
11     return d[m - 1] + (isLeap(y) && m == 2);
12 }
13
14 int getDay(int y, int m, int d) {
15     int ans = 0;
16     for (int i = 1970; i < y; i++) {
17         ans += 365 + isLeap(i);
18     }
19     for (int i = 1; i < m; i++) {
20         ans += daysInMonth(y, i);
21     }
22     ans += d;
23     return (ans + 2) % 7 + 1;
24 }

```

6.19 03 - 欧拉筛

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

```

1 /** 欧拉筛
2 * 2023-11-14: https://qoj.ac/submission/251234
3 **/
4 std::vector<int> minp, primes;

```

```

5
6 void sieve(int n) {
7     minp.assign(n + 1, 0);
8     primes.clear();
9
10    for (int i = 2; i <= n; i++) {
11        if (minp[i] == 0) {
12            minp[i] = i;
13            primes.push_back(i);
14        }
15
16        for (auto p : primes) {
17            if (i * p > n) {
18                break;
19            }
20            minp[i * p] = p;
21            if (p == minp[i]) {
22                break;
23            }
24        }
25    }
26 }
27
28 bool isprime(int n) {
29     return minp[n] == n;
30 }

```

6.20 04 - 莫比乌斯函数筛 (莫比乌斯反演)

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

```

1  /** 莫比乌斯函数筛 (莫比乌斯函数/反演)
2   *   2023-03-04: https://atcoder.jp/contests/tupc2022/submissions/39391116
3   **/
4  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.hpp**

```

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;
8      }
9      int g = exgcd(b, a % b, y, x);
10     y -= a / b * x;
11     return g;
12 }
13
14 /** 扩展欧几里得 (exgcd)
15 *   2023-09-05: https://qoj.ac/submission/165983
16 **/
17 std::array<i64, 3> exgcd(i64 a, i64 b) {
18     if (!b) {
19         return {a, 1, 0};
20     }
21     auto [g, x, y] = exgcd(b, a % b);
22     return {g, y, x - a / b * y};
23 }

```

6.22 06A - 欧拉函数 (求解单个数的欧拉函数)

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

```

1  /** 欧拉函数 (求解单个数的欧拉函数)
2   *   2023-10-09: https://atcoder.jp/contests/tenka1-2017/submissions/46411797
3   **/
4  int phi(int n) {
5      int res = n;
6      for (int i = 2; i * i <= n; i++) {
7          if (n % i == 0) {

```

```

8         while (n % i == 0) {
9             n /= i;
10        }
11        res = res / i * (i - 1);
12    }
13 }
14 if (n > 1) {
15     res = res / n * (n - 1);
16 }
17 return res;
18 }

```

6.23 06B - 欧拉函数 (求解全部数的欧拉函数)

Listing 54: math/06B-Phi-Sieve.hpp

```

1  /** 欧拉函数 (求解全部数的欧拉函数)
2   *   2023-09-24: https://qoj.ac/submission/187055
3   **/
4  constexpr int N = 1E7;
5  constexpr int P = 1000003;
6
7  bool isprime[N + 1];
8  int phi[N + 1];
9  std::vector<int> primes;
10
11 void sieve(void) {
12     std::fill(isprime + 2, isprime + N + 1, true);
13     phi[1] = 1;
14     for (int i = 2; i <= N; i++) {
15         if (isprime[i]) {
16             primes.push_back(i);
17             phi[i] = i - 1;
18         }
19         for (auto p : primes) {
20             if (i * p > N) {
21                 break;
22             }
23             isprime[i * p] = false;
24             if (i % p == 0) {
25                 phi[i * p] = phi[i] * p;
26                 break;
27             }
28             phi[i * p] = phi[i] * (p - 1);
29         }
30     }
31 }

```

6.24 07A - 组合数 (小范围预处理, 逆元 + 杨辉三角)

Listing 55: math/07A-Comb.hpp

```

1  /** 组合数 (小范围预处理, 逆元+杨辉三角)
2   *   2024-03-14: https://qoj.ac/submission/353877
3   *   2023-10-06: https://qoj.ac/submission/203196
4   **/
5  constexpr int P = 1000000007;
6  constexpr int L = 10000;
7
8  int fac[L + 1], invfac[L + 1];
9  int sumbinom[L + 1][7];
10
11 int binom(int n, int m) {

```

```

12     if (n < m || m < 0) {
13         return 0;
14     }
15     return 1LL * fac[n] * invfac[m] % P * invfac[n - m] % P;
16 }
17
18 int power(int a, int b) {
19     int res = 1;
20     for (; b; b /= 2, a = 1LL * a * a % P) {
21         if (b % 2) {
22             res = 1LL * res * a % P;
23         }
24     }
25     return res;
26 }
27
28 int main() {
29     fac[0] = 1;
30     for (int i = 1; i <= L; i++) {
31         fac[i] = 1LL * fac[i - 1] * i % P;
32     }
33     invfac[L] = power(fac[L], P - 2);
34     for (int i = L; i; i--) {
35         invfac[i - 1] = 1LL * invfac[i] * i % P;
36     }
37
38     sumbinom[0][0] = 1;
39     for (int i = 1; i <= L; i++) {
40         for (int j = 0; j < 7; j++) {
41             sumbinom[i][j] = (sumbinom[i - 1][j] + sumbinom[i - 1][(j + 6) % 7]) % P;
42         }
43     }
44 }

```

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

Listing 56: math/07B-Comb.hpp

```

1  /** 组合数 (Comb, with. MInt & MLong)
2  *   2023-08-26: https://codeforces.com/contest/1864/submission/220584872
3  **/
4  struct Comb {
5      int n;
6      std::vector<Z> _fac;
7      std::vector<Z> _invfac;
8      std::vector<Z> _inv;
9
10     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
11     Comb(int n) : Comb() {
12         init(n);
13     }
14
15     void init(int m) {
16         m = std::min(m, Z::getMod() - 1);
17         if (m <= n) return;
18         _fac.resize(m + 1);
19         _invfac.resize(m + 1);
20         _inv.resize(m + 1);
21
22         for (int i = n + 1; i <= m; i++) {
23             _fac[i] = _fac[i - 1] * i;
24         }
25         _invfac[m] = _fac[m].inv();
26         for (int i = m; i > n; i--) {

```

```

27     _invfac[i - 1] = _invfac[i] * i;
28     _inv[i] = _invfac[i] * _fac[i - 1];
29 }
30 n = m;
31 }
32
33 Z fac(int m) {
34     if (m > n) init(2 * m);
35     return _fac[m];
36 }
37 Z invfac(int m) {
38     if (m > n) init(2 * m);
39     return _invfac[m];
40 }
41 Z inv(int m) {
42     if (m > n) init(2 * m);
43     return _inv[m];
44 }
45 Z binom(int n, int m) {
46     if (n < m || m < 0) return 0;
47     return fac(n) * invfac(m) * invfac(n - m);
48 }
49 } comb;

```

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

Listing 57: math/08-Prime.hpp

```

1  i64 mul(i64 a, i64 b, i64 m) {
2      return static_cast<__int128>(a) * b % m;
3  }
4  i64 power(i64 a, i64 b, i64 m) {
5      i64 res = 1 % m;
6      for (; b >>= 1; a = mul(a, a, m))
7          if (b & 1)
8              res = mul(res, a, m);
9      return res;
10 }
11 bool isprime(i64 n) {
12     if (n < 2)
13         return false;
14     static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
15     int s = __builtin_ctzll(n - 1);
16     i64 d = (n - 1) >> s;
17     for (auto a : A) {
18         if (a == n)
19             return true;
20         i64 x = power(a, d, n);
21         if (x == 1 || x == n - 1)
22             continue;
23         bool ok = false;
24         for (int i = 0; i < s - 1; ++i) {
25             x = mul(x, x, n);
26             if (x == n - 1) {
27                 ok = true;
28                 break;
29             }
30         }
31         if (!ok)
32             return false;
33     }
34     return true;
35 }

```

```

36 std::vector<i64> factorize(i64 n) {
37     std::vector<i64> p;
38     std::function<void(i64)> f = [&](i64 n) {
39         if (n <= 10000) {
40             for (int i = 2; i * i <= n; ++i)
41                 for (; n % i == 0; n /= i)
42                     p.push_back(i);
43             if (n > 1)
44                 p.push_back(n);
45             return;
46         }
47         if (isprime(n)) {
48             p.push_back(n);
49             return;
50         }
51         auto g = [&](i64 x) {
52             return (mul(x, x, n) + 1) % n;
53         };
54         i64 x0 = 2;
55         while (true) {
56             i64 x = x0;
57             i64 y = x0;
58             i64 d = 1;
59             i64 power = 1, lam = 0;
60             i64 v = 1;
61             while (d == 1) {
62                 y = g(y);
63                 ++lam;
64                 v = mul(v, std::abs(x - y), n);
65                 if (lam % 127 == 0) {
66                     d = std::gcd(v, n);
67                     v = 1;
68                 }
69                 if (power == lam) {
70                     x = y;
71                     power *= 2;
72                     lam = 0;
73                     d = std::gcd(v, n);
74                     v = 1;
75                 }
76             }
77             if (d != n) {
78                 f(d);
79                 f(n / d);
80                 return;
81             }
82             ++x0;
83         }
84     };
85     f(n);
86     std::sort(p.begin(), p.end());
87     return p;
88 }

```

6.27 09A - 平面几何 (Point)

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

```

1 /** 平面几何 (Point)
2  *   2023-09-22: https://qoj.ac/submission/185408
3  **/
4 template<class T>
5 struct Point {

```

```

6     T x;
7     T y;
8     Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}
9
10    template<class U>
11    operator Point<U>() {
12        return Point<U>(U(x), U(y));
13    }
14    Point &operator+=(const Point &p) & {
15        x += p.x;
16        y += p.y;
17        return *this;
18    }
19    Point &operator-=(const Point &p) & {
20        x -= p.x;
21        y -= p.y;
22        return *this;
23    }
24    Point &operator*=(const T &v) & {
25        x *= v;
26        y *= v;
27        return *this;
28    }
29    Point &operator/=(const T &v) & {
30        x /= v;
31        y /= v;
32        return *this;
33    }
34    Point operator-() const {
35        return Point(-x, -y);
36    }
37    friend Point operator+(Point a, const Point &b) {
38        return a += b;
39    }
40    friend Point operator-(Point a, const Point &b) {
41        return a -= b;
42    }
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;
170         }
171     }
172
173     return t == 1;
174 }
175
176 // 0 : not intersect
177 // 1 : strictly intersect
178 // 2 : overlap
179 // 3 : intersect at endpoint
180 template<class T>
181 std::tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1, const Line<T> &l2) {
182     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
183         return {0, Point<T>(), Point<T>()};
184     }
185     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
186         return {0, Point<T>(), Point<T>()};
187     }
188     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
189         return {0, Point<T>(), Point<T>()};
190     }
191     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
192         return {0, Point<T>(), Point<T>()};
193     }
194     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
195         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
196             return {0, Point<T>(), Point<T>()};

```



```

197     } else {
198         auto maxx1 = std::max(l1.a.x, l1.b.x);
199         auto minx1 = std::min(l1.a.x, l1.b.x);
200         auto maxy1 = std::max(l1.a.y, l1.b.y);
201         auto miny1 = std::min(l1.a.y, l1.b.y);
202         auto maxx2 = std::max(l2.a.x, l2.b.x);
203         auto minx2 = std::min(l2.a.x, l2.b.x);
204         auto maxy2 = std::max(l2.a.y, l2.b.y);
205         auto miny2 = std::min(l2.a.y, l2.b.y);
206         Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
207         Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
208         if (!pointOnSegment(p1, l1)) {
209             std::swap(p1.y, p2.y);
210         }
211         if (p1 == p2) {
212             return {3, p1, p2};
213         } else {
214             return {2, p1, p2};
215         }
216     }
217 }
218 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
219 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
220 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
221 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
222
223 if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0) || (cp3 < 0 && cp4 < 0)) {
224     return {0, Point<T>(), Point<T>()};
225 }
226
227 Point p = lineIntersection(l1, l2);
228 if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
229     return {1, p, p};
230 } else {
231     return {3, p, p};
232 }
233 }
234
235 template<class T>
236 double distanceSS(const Line<T> &l1, const Line<T> &l2) {
237     if (std::get<0>(segmentIntersection(l1, l2)) != 0) {
238         return 0.0;
239     }
240     return std::min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(l2.a, l1), distancePS(l2.b, l1)});
241 }
242
243 template<class T>
244 bool segmentInPolygon(const Line<T> &l, const std::vector<Point<T>> &p) {
245     int n = p.size();
246     if (!pointInPolygon(l.a, p)) {
247         return false;
248     }
249     if (!pointInPolygon(l.b, p)) {
250         return false;
251     }
252     for (int i = 0; i < n; i++) {
253         auto u = p[i];
254         auto v = p[(i + 1) % n];
255         auto w = p[(i + 2) % n];
256         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
257
258         if (t == 1) {

```

```

259         return false;
260     }
261     if (t == 0) {
262         continue;
263     }
264     if (t == 2) {
265         if (pointOnSegment(v, l) && v != l.a && v != l.b) {
266             if (cross(v - u, w - v) > 0) {
267                 return false;
268             }
269         }
270     } else {
271         if (p1 != u && p1 != v) {
272             if (pointOnLineLeft(l.a, Line(v, u))
273                 || pointOnLineLeft(l.b, Line(v, u))) {
274                 return false;
275             }
276         } else if (p1 == v) {
277             if (l.a == v) {
278                 if (pointOnLineLeft(u, l)) {
279                     if (pointOnLineLeft(w, l)
280                         && pointOnLineLeft(w, Line(u, v))) {
281                         return false;
282                     }
283                 } else {
284                     if (pointOnLineLeft(w, l)
285                         || pointOnLineLeft(w, Line(u, v))) {
286                         return false;
287                     }
288                 }
289             } else if (l.b == v) {
290                 if (pointOnLineLeft(u, Line(l.b, l.a))) {
291                     if (pointOnLineLeft(w, Line(l.b, l.a))
292                         && pointOnLineLeft(w, Line(u, v))) {
293                         return false;
294                     }
295                 } else {
296                     if (pointOnLineLeft(w, Line(l.b, l.a))
297                         || pointOnLineLeft(w, Line(u, v))) {
298                         return false;
299                     }
300                 }
301             } else {
302                 if (pointOnLineLeft(u, l)) {
303                     if (pointOnLineLeft(w, Line(l.b, l.a))
304                         || pointOnLineLeft(w, Line(u, v))) {
305                         return false;
306                     }
307                 } else {
308                     if (pointOnLineLeft(w, l)
309                         || pointOnLineLeft(w, Line(u, v))) {
310                         return false;
311                     }
312                 }
313             }
314         }
315     }
316 }
317 return true;
318 }
319
320 template<class T>
321 std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
322     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {

```

```

323     auto d1 = l1.b - l1.a;
324     auto d2 = l2.b - l2.a;
325
326     if (sgn(d1) != sgn(d2)) {
327         return sgn(d1) == 1;
328     }
329
330     return cross(d1, d2) > 0;
331 });
332
333 std::deque<Line<T>> ls;
334 std::deque<Point<T>> ps;
335 for (auto l : lines) {
336     if (ls.empty()) {
337         ls.push_back(l);
338         continue;
339     }
340
341     while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
342         ps.pop_back();
343         ls.pop_back();
344     }
345
346     while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
347         ps.pop_front();
348         ls.pop_front();
349     }
350
351     if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
352         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
353
354             if (!pointOnLineLeft(ls.back().a, l)) {
355                 assert(ls.size() == 1);
356                 ls[0] = l;
357             }
358             continue;
359         }
360         return {};
361     }
362
363     ps.push_back(lineIntersection(ls.back(), l));
364     ls.push_back(l);
365 }
366
367 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
368     ps.pop_back();
369     ls.pop_back();
370 }
371 if (ls.size() <= 2) {
372     return {};
373 }
374 ps.push_back(lineIntersection(ls[0], ls.back()));
375
376 return std::vector(ps.begin(), ps.end());
377 }
378
379 using real = long double;
380 using P = Point<real>;
381
382 constexpr real eps = 0;

```

6.28 09B - 平面几何 (with. std::complex)

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

```

1  /** 平面几何 (with. std::complex)
2   *   2023-09-04: https://qoj.ac/submission/164445
3   **/
4  using Point = std::complex<long double>;
5
6  #define x real
7  #define y imag
8
9  long double dot(const Point &a, const Point &b) {
10     return (std::conj(a) * b).x();
11 }
12
13 long double cross(const Point &a, const Point &b) {
14     return (std::conj(a) * b).y();
15 }
16
17 long double length(const Point &a) {
18     return std::sqrt(dot(a, a));
19 }
20
21 long double dist(const Point &a, const Point &b) {
22     return length(a - b);
23 }
24
25 long double get(const Point &a, const Point &b, const Point &c, const Point &d) {
26     auto e = a + (b - a) * cross(c - a, d - a) / cross(b - a, d - c);
27     return dist(d, e);
28 }

```

6.29 10 - 立体几何 (Point)

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

```

1  /** 立体几何
2   *   2023-09-25 (i64): https://qoj.ac/submission/188519
3   *   2023-09-28 (double): https://qoj.ac/submission/190463
4   **/
5  using i64 = long long;
6  using real = double;
7
8  struct Point {
9     real x = 0;
10    real y = 0;
11    real z = 0;
12 };
13
14 Point operator+(const Point &a, const Point &b) {
15     return {a.x + b.x, a.y + b.y, a.z + b.z};
16 }
17
18 Point operator-(const Point &a, const Point &b) {
19     return {a.x - b.x, a.y - b.y, a.z - b.z};
20 }
21
22 Point operator*(const Point &a, real b) {
23     return {a.x * b, a.y * b, a.z * b};
24 }
25
26 Point operator/(const Point &a, real b) {
27     return {a.x / b, a.y / b, a.z / b};
28 }
29
30 real length(const Point &a) {

```

```

31     return std::hypot(a.x, a.y, a.z);
32 }
33
34 Point normalize(const Point &a) {
35     real l = length(a);
36     return {a.x / l, a.y / l, a.z / l};
37 }
38
39 real getAng(real a, real b, real c) {
40     return std::acos((a * a + b * b - c * c) / 2 / a / b);
41 }
42
43 std::ostream &operator<<(std::ostream &os, const Point &a) {
44     return os << "(" << a.x << ", " << a.y << ", " << a.z << ")";
45 }
46
47 real dot(const Point &a, const Point &b) {
48     return a.x * b.x + a.y * b.y + a.z * b.z;
49 }
50
51 Point cross(const Point &a, const Point &b) {
52     return {
53         a.y * b.z - a.z * b.y,
54         a.z * b.x - a.x * b.z,
55         a.x * b.y - a.y * b.x
56     };
57 }

```

6.30 11A - 静态凸包 (with. Point, 旧版)

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

```

1  /** 静态凸包 (with. Point, 旧版)
2  *   2023-04-09: https://cf.dianhsu.com/gym/104288/submission/201412835
3  **/
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.hpp**

```

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
52 const double inf = INFINITY;

```

6.32 11C - 静态凸包 (with. std::complex)

Listing 63: math/11C-Convex-Hull.hpp

```

1  /** 静态凸包 (with. std::complex)
2   *   2022-02-04: https://loj.ac/s/1370861
3   **/
4  using Point = std::complex<i64>;
5
6  #define x real
7  #define y imag
8
9  auto dot(const Point &a, const Point &b) {
10     return (std::conj(a) * b).x();
11 }
12
13 auto cross(const Point &a, const Point &b) {
14     return (std::conj(a) * b).y();
15 }
16
17 auto rot(const Point &p) {
18     return Point(-p.y(), p.x());
19 }
20
21 auto complexHull(std::vector<Point> a) {
22     std::sort(a.begin(), a.end(), [&](auto a, auto b) {

```

```

23     if (a.x() != b.x()) {
24         return a.x() < b.x();
25     } else {
26         return a.y() < b.y();
27     }
28 };
29
30 std::vector<Point> l, h;
31
32 for (auto p : a) {
33     while (l.size() > 1 && cross(l.back() - l[l.size() - 2], p - l.back()) <= 0) {
34         l.pop_back();
35     }
36
37     while (h.size() > 1 && cross(h.back() - h[h.size() - 2], p - h.back()) >= 0) {
38         h.pop_back();
39     }
40
41     l.push_back(p);
42     h.push_back(p);
43 }
44
45 std::reverse(h.begin(), h.end());
46
47 h.insert(h.end(), l.begin() + 1, l.end() - 1);
48
49 return h;
50 }
51
52 int sgn(Point p) {
53     if (p.y() > 0 || (p.y() == 0 && p.x() < 0)) {
54         return 0;
55     } else {
56         return 1;
57     }
58 }

```

6.33 12A - 多项式 (Poly, with. Z)

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

```

1  /** 多项式相关 (Poly, with. Z)
2   *   2023-02-06: https://atcoder.jp/contests/arc155/submissions/38664055
3   **/
4  std::vector<int> rev;
5  std::vector<Z> roots{0, 1};
6  void dft(std::vector<Z> &a) {
7      int n = a.size();
8
9      if (int(rev.size()) != n) {
10         int k = __builtin_ctz(n) - 1;
11         rev.resize(n);
12         for (int i = 0; i < n; i++) {
13             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
14         }
15     }
16
17     for (int i = 0; i < n; i++) {
18         if (rev[i] < i) {
19             std::swap(a[i], a[rev[i]]);
20         }
21     }
22     if (int(roots.size()) < n) {

```



```

23     int k = __builtin_ctz(roots.size());
24     roots.resize(n);
25     while ((1 << k) < n) {
26         Z e = power(Z(3), (P - 1) >> (k + 1));
27         for (int i = 1 << (k - 1); i < (1 << k); i++) {
28             roots[2 * i] = roots[i];
29             roots[2 * i + 1] = roots[i] * e;
30         }
31         k++;
32     }
33 }
34 for (int k = 1; k < n; k *= 2) {
35     for (int i = 0; i < n; i += 2 * k) {
36         for (int j = 0; j < k; j++) {
37             Z u = a[i + j];
38             Z v = a[i + j + k] * roots[k + j];
39             a[i + j] = u + v;
40             a[i + j + k] = u - v;
41         }
42     }
43 }
44 }
45 void idft(std::vector<Z> &a) {
46     int n = a.size();
47     std::reverse(a.begin() + 1, a.end());
48     dft(a);
49     Z inv = (1 - P) / n;
50     for (int i = 0; i < n; i++) {
51         a[i] *= inv;
52     }
53 }
54 struct Poly {
55     std::vector<Z> a;
56     Poly() {}
57     explicit Poly(int size, std::function<Z(int)> f = [](int) { return 0; }) : a(size) {
58         for (int i = 0; i < size; i++) {
59             a[i] = f(i);
60         }
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 {
250             int m = (l + r) / 2;
251             build(2 * p, l, m);
252             build(2 * p + 1, m, r);
253             q[p] = q[2 * p] * q[2 * p + 1];
254         }
255     };
256     build(1, 0, n);
257     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
258         if (r - l == 1) {
259             if (l < int(ans.size())) {
260                 ans[l] = num[0];
261             }
262         } else {
263             int m = (l + r) / 2;
264             work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
265             work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
266         }
267     };
268     work(1, 0, n, mulT(q[1].inv(n)));
269     return ans;
270 }
271 };

```

6.34 12B - 多项式 (Poly, with. MInt)

Listing 65: math/12B-Poly.hpp

```

1  /** 多项式相关 (Poly, with. MInt & MLong)
2   *   2023-09-20: https://atcoder.jp/contests/arc163/submissions/45737810
3   **/
4  std::vector<int> rev;
5  template<int P>
6  std::vector<MInt<P>> roots{0, 1};
7
8  template<int P>
9  constexpr MInt<P> findPrimitiveRoot() {
10     MInt<P> i = 2;
11     int k = __builtin_ctz(P - 1);
12     while (true) {
13         if (power(i, (P - 1) / 2) != 1) {
14             break;
15         }
16         i += 1;
17     }
18     return power(i, (P - 1) >> k);
19 }
20
21 template<int P>
22 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
23
24 template<>
25 constexpr MInt<998244353> primitiveRoot<998244353> {31};
26
27 template<int P>
28 constexpr void dft(std::vector<MInt<P>> &a) {
29     int n = a.size();
30
31     if (int(rev.size()) != n) {
32         int k = __builtin_ctz(n) - 1;
33         rev.resize(n);
34         for (int i = 0; i < n; i++) {
35             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
36         }
37     }
38
39     for (int i = 0; i < n; i++) {
40         if (rev[i] < i) {
41             std::swap(a[i], a[rev[i]]);
42         }
43     }
44     if (roots<P>.size() < n) {
45         int k = __builtin_ctz(roots<P>.size());
46         roots<P>.resize(n);
47         while ((1 << k) < n) {
48             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
49             for (int i = 1 << (k - 1); i < (1 << k); i++) {
50                 roots<P>[2 * i] = roots<P>[i];
51                 roots<P>[2 * i + 1] = roots<P>[i] * e;
52             }
53             k++;
54         }
55     }
56     for (int k = 1; k < n; k *= 2) {
57         for (int i = 0; i < n; i += 2 * k) {
58             for (int j = 0; j < k; j++) {
59                 MInt<P> u = a[i + j];
60                 MInt<P> v = a[i + j + k] * roots<P>[k + j];

```

```

61         a[i + j] = u + v;
62         a[i + j + k] = u - v;
63     }
64 }
65 }
66 }
67
68 template<int P>
69 constexpr void idft(std::vector<MInt<P>> &a) {
70     int n = a.size();
71     std::reverse(a.begin() + 1, a.end());
72     dft(a);
73     MInt<P> inv = (1 - P) / n;
74     for (int i = 0; i < n; i++) {
75         a[i] *= inv;
76     }
77 }
78
79 template<int P = 998244353>
80 struct Poly : public std::vector<MInt<P>> {
81     using Value = MInt<P>;
82
83     Poly() : std::vector<Value>() {}
84     explicit constexpr Poly(int n) : std::vector<Value>(n) {}
85
86     explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a) {}
87     constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(a) {}
88
89     template<class InputIt, class = std::::RequireInputIter<InputIt>>
90     explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(first, last) {}
91
92     template<class F>
93     explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
94         for (int i = 0; i < n; i++) {
95             (*this)[i] = f(i);
96         }
97     }
98
99     constexpr Poly shift(int k) const {
100         if (k >= 0) {
101             auto b = *this;
102             b.insert(b.begin(), k, 0);
103             return b;
104         } else if (this->size() <= -k) {
105             return Poly();
106         } else {
107             return Poly(this->begin() + (-k), this->end());
108         }
109     }
110
111     constexpr Poly trunc(int k) const {
112         Poly f = *this;
113         f.resize(k);
114         return f;
115     }
116
117     constexpr friend Poly operator+(const Poly &a, const Poly &b) {
118         Poly res(std::max(a.size(), b.size()));
119         for (int i = 0; i < a.size(); i++) {
120             res[i] += a[i];
121         }
122         for (int i = 0; i < b.size(); i++) {
123             res[i] += b[i];
124         }
125         return res;
126     }

```

```

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

```

```

188     }
189     return a;
190 }
191 constexpr Poly &operator+=(Poly b) {
192     return (*this) = (*this) + b;
193 }
194 constexpr Poly &operator-=(Poly b) {
195     return (*this) = (*this) - b;
196 }
197 constexpr Poly &operator*=(Poly b) {
198     return (*this) = (*this) * b;
199 }
200 constexpr Poly &operator*=(Value b) {
201     return (*this) = (*this) * b;
202 }
203 constexpr Poly &operator/=(Value b) {
204     return (*this) = (*this) / b;
205 }
206 constexpr Poly deriv() const {
207     if (this->empty()) {
208         return Poly();
209     }
210     Poly res(this->size() - 1);
211     for (int i = 0; i < this->size() - 1; ++i) {
212         res[i] = (i + 1) * (*this)[i + 1];
213     }
214     return res;
215 }
216 constexpr Poly integr() const {
217     Poly res(this->size() + 1);
218     for (int i = 0; i < this->size(); ++i) {
219         res[i + 1] = (*this)[i] / (i + 1);
220     }
221     return res;
222 }
223 constexpr Poly inv(int m) const {
224     Poly x((*this)[0].inv());
225     int k = 1;
226     while (k < m) {
227         k *= 2;
228         x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
229     }
230     return x.trunc(m);
231 }
232 constexpr Poly log(int m) const {
233     return (deriv() * inv(m)).integr().trunc(m);
234 }
235 constexpr Poly exp(int m) const {
236     Poly x{1};
237     int k = 1;
238     while (k < m) {
239         k *= 2;
240         x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
241     }
242     return x.trunc(m);
243 }
244 constexpr Poly pow(int k, int m) const {
245     int i = 0;
246     while (i < this->size() && (*this)[i] == 0) {
247         i++;
248     }
249     if (i == this->size() || 1LL * i * k >= m) {
250         return Poly(m);

```



```

251     }
252     Value v = (*this)[i];
253     auto f = shift(-i) * v.inv();
254     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
255 }
256 constexpr Poly sqrt(int m) const {
257     Poly x{1};
258     int k = 1;
259     while (k < m) {
260         k *= 2;
261         x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
262     }
263     return x.trunc(m);
264 }
265 constexpr Poly mult(Poly b) const {
266     if (b.size() == 0) {
267         return Poly();
268     }
269     int n = b.size();
270     std::reverse(b.begin(), b.end());
271     return ((*this) * b).shift(-(n - 1));
272 }
273 constexpr std::vector<Value> eval(std::vector<Value> x) const {
274     if (this->size() == 0) {
275         return std::vector<Value>(x.size(), 0);
276     }
277     const int n = std::max(x.size(), this->size());
278     std::vector<Poly> q(4 * n);
279     std::vector<Value> ans(x.size());
280     x.resize(n);
281     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
282         if (r - l == 1) {
283             q[p] = Poly{1, -x[l]};
284         } else {
285             int m = (l + r) / 2;
286             build(2 * p, l, m);
287             build(2 * p + 1, m, r);
288             q[p] = q[2 * p] * q[2 * p + 1];
289         }
290     };
291     build(1, 0, n);
292     std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
293         if (r - l == 1) {
294             if (l < int(ans.size())) {
295                 ans[l] = num[0];
296             }
297         } else {
298             int m = (l + r) / 2;
299             work(2 * p, l, m, num.mult(q[2 * p + 1]).resize(m - l));
300             work(2 * p + 1, m, r, num.mult(q[2 * p]).resize(r - m));
301         }
302     };
303     work(1, 0, n, mult(q[1].inv(n)));
304     return ans;
305 }
306 };
307
308 template<int P = 998244353>
309 Poly<P> berlekampMassey(const Poly<P> &s) {
310     Poly<P> c;
311     Poly<P> oldC;
312     int f = -1;
313     for (int i = 0; i < s.size(); i++) {

```

```

314     auto delta = s[i];
315     for (int j = 1; j <= c.size(); j++) {
316         delta -= c[j - 1] * s[i - j];
317     }
318     if (delta == 0) {
319         continue;
320     }
321     if (f == -1) {
322         c.resize(i + 1);
323         f = i;
324     } else {
325         auto d = oldC;
326         d *= -1;
327         d.insert(d.begin(), 1);
328         MInt<P> df1 = 0;
329         for (int j = 1; j <= d.size(); j++) {
330             df1 += d[j - 1] * s[f + 1 - j];
331         }
332         assert(df1 != 0);
333         auto coef = delta / df1;
334         d *= coef;
335         Poly<P> zeros(i - f - 1);
336         zeros.insert(zeros.end(), d.begin(), d.end());
337         d = zeros;
338         auto temp = c;
339         c += d;
340         if (i - temp.size() > f - oldC.size()) {
341             oldC = temp;
342             f = i;
343         }
344     }
345 }
346 c *= -1;
347 c.insert(c.begin(), 1);
348 return c;
349 }
350
351
352 template<int P = 998244353>
353 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
354     int m = q.size() - 1;
355     while (n > 0) {
356         auto newq = q;
357         for (int i = 1; i <= m; i += 2) {
358             newq[i] *= -1;
359         }
360         auto newp = p * newq;
361         newq = q * newq;
362         for (int i = 0; i < m; i++) {
363             p[i] = newp[i * 2 + n % 2];
364         }
365         for (int i = 0; i <= m; i++) {
366             q[i] = newq[i * 2];
367         }
368         n /= 2;
369     }
370     return p[0] / q[0];
371 }
372
373 struct Comb {
374     int n;
375     std::vector<Z> _fac;
376     std::vector<Z> _invfac;
377     std::vector<Z> _inv;
378

```

```

379 Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
380 Comb(int n) : Comb() {
381     init(n);
382 }
383
384 void init(int m) {
385     m = std::min(m, Z::getMod() - 1);
386     if (m <= n) return;
387     _fac.resize(m + 1);
388     _invfac.resize(m + 1);
389     _inv.resize(m + 1);
390
391     for (int i = n + 1; i <= m; i++) {
392         _fac[i] = _fac[i - 1] * i;
393     }
394     _invfac[m] = _fac[m].inv();
395     for (int i = m; i > n; i--) {
396         _invfac[i - 1] = _invfac[i] * i;
397         _inv[i] = _invfac[i] * _fac[i - 1];
398     }
399     n = m;
400 }
401
402 Z fac(int m) {
403     if (m > n) init(2 * m);
404     return _fac[m];
405 }
406 Z invfac(int m) {
407     if (m > n) init(2 * m);
408     return _invfac[m];
409 }
410 Z inv(int m) {
411     if (m > n) init(2 * m);
412     return _inv[m];
413 }
414 Z binom(int n, int m) {
415     if (n < m || m < 0) return 0;
416     return fac(n) * invfac(m) * invfac(n - m);
417 }
418 } comb;
419
420 Poly<P> get(int n, int m) {
421     if (m == 0) {
422         return Poly(n + 1);
423     }
424     if (m % 2 == 1) {
425         auto f = get(n, m - 1);
426         Z p = 1;
427         for (int i = 0; i <= n; i++) {
428             f[n - i] += comb.binom(n, i) * p;
429             p *= m;
430         }
431         return f;
432     }
433     auto f = get(n, m / 2);
434     auto fm = f;
435     for (int i = 0; i <= n; i++) {
436         fm[i] *= comb.fac(i);
437     }
438     Poly pw(n + 1);
439     pw[0] = 1;
440     for (int i = 1; i <= n; i++) {
441         pw[i] = pw[i - 1] * (m / 2);

```

```

442     }
443     for (int i = 0; i <= n; i++) {
444         pw[i] *= comb.invfac(i);
445     }
446     fm = fm.mult(pw);
447     for (int i = 0; i <= n; i++) {
448         fm[i] *= comb.invfac(i);
449     }
450     return f + fm;
451 }

```

6.35 12C - 多项式乘法

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

```

1  /** 多项式乘法
2   *   2024-03-10: https://qoj.ac/submission/350298
3   **/
4  constexpr int P = 998244353;
5
6  int power(int a, int b) {
7      int res = 1;
8      for (; b; b /= 2, a = 1LL * a * a % P) {
9          if (b % 2) {
10             res = 1LL * res * a % P;
11         }
12     }
13     return res;
14 }
15
16 std::vector<int> rev, roots {0, 1};
17
18 void dft(std::vector<int> &a) {
19     int n = a.size();
20     if (int(rev.size()) != n) {
21         int k = __builtin_ctz(n) - 1;
22         rev.resize(n);
23         for (int i = 0; i < n; i++) {
24             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
25         }
26     }
27     for (int i = 0; i < n; i++) {
28         if (rev[i] < i) {
29             std::swap(a[i], a[rev[i]]);
30         }
31     }
32     if (roots.size() < n) {
33         int k = __builtin_ctz(roots.size());
34         roots.resize(n);
35         while ((1 << k) < n) {
36             int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));
37             for (int i = 1 << (k - 1); i < (1 << k); i++) {
38                 roots[2 * i] = roots[i];
39                 roots[2 * i + 1] = 1LL * roots[i] * e % P;
40             }
41             k++;
42         }
43     }
44
45     for (int k = 1; k < n; k *= 2) {
46         for (int i = 0; i < n; i += 2 * k) {
47             for (int j = 0; j < k; j++) {
48                 int u = a[i + j];

```

```

49         int v = 1LL * a[i + j + k] * roots[k + j] % P;
50         a[i + j] = (u + v) % P;
51         a[i + j + k] = (u - v) % P;
52     }
53 }
54 }
55 }
56
57 void idft(std::vector<int> &a) {
58     int n = a.size();
59     std::reverse(a.begin() + 1, a.end());
60     dft(a);
61     int inv = (1 - P) / n;
62     for (int i = 0; i < n; i++) {
63         a[i] = 1LL * a[i] * inv % P;
64     }
65 }
66
67 std::vector<int> mul(std::vector<int> a, std::vector<int> b) {
68     int n = 1, tot = a.size() + b.size() - 1;
69     while (n < tot) {
70         n *= 2;
71     }
72     if (tot < 128) {
73         std::vector<int> c(a.size() + b.size() - 1);
74         for (int i = 0; i < a.size(); i++) {
75             for (int j = 0; j < b.size(); j++) {
76                 c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % P;
77             }
78         }
79         return c;
80     }
81     a.resize(n);
82     b.resize(n);
83     dft(a);
84     dft(b);
85     for (int i = 0; i < n; i++) {
86         a[i] = 1LL * a[i] * b[i] % P;
87     }
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.hpp

```

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.hpp`

```

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;
6     for (; b; b /= 2, a = 1LL * a * a % p) {
7         if (b % 2) {
8             res = 1LL * res * a % p;
9         }
10    }
11    return res;
12 }
13
14 int qint(int n, int q, int p) {
15     return 1LL * (power(q, n, p) - 1) * power(q - 1, p - 2, p) % p;
16 }
17
18 int qBinomial(int n, int k, int q, int p) {
19     if (q == 0) {
20         return 1;
21     }
22     int r = 0;
23     int x = 1;
24     do {
25         x = 1LL * x * q % p;
26         r++;
27     } while (x != 1);
28
29     if (n / r > k / r + (n - k) / r) {
30         return 0;

```

```

31     }
32     int num = 1, den = 1;
33     for (int i = 1; i <= k % r; i++) {
34         num = 1LL * num * qint(n % r - i + 1, q, p) % p;
35         den = 1LL * den * qint(i, q, p) % p;
36     }
37     n /= r, k /= r;
38     while (n > 0 || k > 0) {
39         if (n % p < k % p) {
40             return 0;
41         }
42         for (int i = 1; i <= k % p; i++) {
43             num = 1LL * num * (n % p - i + 1) % p;
44             den = 1LL * den * i % p;
45         }
46         n /= p, k /= p;
47     }
48     int ans = 1LL * num * power(den, p - 2, p) % p;
49     return ans;
50 }

```

6.38 13C - 生成函数 (Binomial 任意模数二项式)

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

```

1  /** 生成函数 (Binomial 任意模数二项式)
2   *   2023-08-22: https://codeforces.com/contest/896/submission/219861532
3   **/
4  std::vector<std::pair<int, int>> factorize(int n) {
5      std::vector<std::pair<int, int>> factors;
6      for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
7          if (n % i == 0) {
8              int t = 0;
9              for (; n % i == 0; n /= i)
10                 ++t;
11             factors.emplace_back(i, t);
12         }
13     }
14     if (n > 1)
15         factors.emplace_back(n, 1);
16     return factors;
17 }
18 constexpr int power(int base, i64 exp) {
19     int res = 1;
20     for (; exp > 0; base *= base, exp /= 2) {
21         if (exp % 2 == 1) {
22             res *= base;
23         }
24     }
25     return res;
26 }
27 constexpr int power(int base, i64 exp, int mod) {
28     int res = 1 % mod;
29     for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
30         if (exp % 2 == 1) {
31             res = 1LL * res * base % mod;
32         }
33     }
34     return res;
35 }
36 int inverse(int a, int m) {
37     int g = m, r = a, x = 0, y = 1;
38     while (r != 0) {

```

```

39     int q = g / r;
40     g %= r;
41     std::swap(g, r);
42     x -= q * y;
43     std::swap(x, y);
44 }
45 return x < 0 ? x + m : x;
46 }
47 int solveModuloEquations(const std::vector<std::pair<int, int>> &e) {
48     int m = 1;
49     for (std::size_t i = 0; i < e.size(); i++) {
50         m *= e[i].first;
51     }
52     int res = 0;
53     for (std::size_t i = 0; i < e.size(); i++) {
54         int p = e[i].first;
55         res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
56     }
57     return res;
58 }
59 constexpr int N = 1E5;
60 class Binomial {
61     const int mod;
62 private:
63     const std::vector<std::pair<int, int>> factors;
64     std::vector<int> pk;
65     std::vector<std::vector<int>> prod;
66     static constexpr i64 exponent(i64 n, int p) {
67         i64 res = 0;
68         for (n /= p; n > 0; n /= p) {
69             res += n;
70         }
71         return res;
72     }
73     int product(i64 n, std::size_t i) {
74         int res = 1;
75         int p = factors[i].first;
76         for (; n > 0; n /= p) {
77             res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) % pk[i] * prod[i][n % pk[i]] % pk[i];
78         }
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)
116                 % pk[i];
117             ans[i] = std::make_pair(pk[i], res);
118         }
119     }
120     return solveModuloEquations(ans);
121 }
};

```

6.39 14 - 自适应辛普森法 Simpson

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

```

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) {
15     double mid = (l + r) / 2;
16     double sl = simpson(l, mid);
17     double sr = simpson(mid, r);
18     if (std::abs(sl + sr - st) <= 15 * eps)
19         return sl + sr + (sl + sr - st) / 15;
20     return integral(l, mid, eps / 2, sl) + integral(mid, r, eps / 2, sr);
21 }
22 double integral(double l, double r) {
23     return integral(l, r, EPS, simpson(l, r));
24 }

```

6.40 15 - 矩阵 (Matrix)

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

```

1  /** 矩阵 (Matrix)
2   *   2024-03-14: https://qoj.ac/submission/353771
3   **/
4  using i64 = long long;
5  using u64 = unsigned long long;
6
7  using Matrix = std::array<u64, 65>;
8

```

```

9 Matrix operator*(const Matrix &a, const Matrix &b) {
10     Matrix c{};
11     for (int i = 0; i <= 64; i++) {
12         for (int j = 0; j <= 64; j++) {
13             if (j == 64 ? i == 64 : (a[i] >> j & 1)) {
14                 c[i] ^= b[j];
15             }
16         }
17     }
18     return c;
19 }
20
21 u64 operator*(u64 a, const Matrix &b) {
22     u64 c = 0;
23     for (int i = 0; i <= 64; i++) {
24         if (i == 64 || (a >> i & 1)) {
25             c ^= b[i];
26         }
27     }
28     return c;
29 }
30
31 Matrix readMatrix() {
32     int m;
33     std::cin >> m;
34
35     Matrix f{};
36     for (int i = 0; i < m; i++) {
37         int s, o;
38         u64 A;
39         std::cin >> s >> o >> A;
40
41         if (o == 0) {
42             for (int j = 0; j < 64; j++) {
43                 if (A >> ((j + s) % 64) & 1) {
44                     f[64] ^= 1ULL << ((j + s) % 64);
45                 } else {
46                     f[j] ^= 1ULL << ((j + s) % 64);
47                 }
48             }
49         } else {
50             for (int j = 0; j < 64; j++) {
51                 if (A >> ((j + s) % 64) & 1) {
52                     f[j] ^= 1ULL << ((j + s) % 64);
53                 }
54             }
55         }
56     }
57
58     u64 B;
59     std::cin >> B;
60     f[64] ^= B;
61
62     return f;
63 }

```

6.41 16 - 高斯消元 (guess)【久远】

Listing 72: math/16-Gauss-Elimination.hpp

```

1 /** 高斯消元 (guess)【久远】
2  * 2020-12-02: https://www.codechef.com/viewsolution/39942900
3  **/
4 std::vector<double> gauss(std::vector<std::vector<double>> a, std::vector<double> b) {

```

```

5     int n = a.size();
6     for (int i = 0; i < n; ++i) {
7         double x = a[i][i];
8         for (int j = i; j < n; ++j) a[i][j] /= x;
9         b[i] /= x;
10        for (int j = 0; j < n; ++j) {
11            if (i == j) continue;
12            x = a[j][i];
13            for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
14            b[j] -= b[i] * x;
15        }
16    }
17    return b;
18 }

```

6.42 01A - 树状数组 (Fenwick 旧版)

Listing 73: ds/01A-Fenwick.hpp

```

1  /** 树状数组 (Fenwick 旧版)
2   *   2023-08-11: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=63382128
3   **/
4  template <typename T>
5  struct Fenwick {
6      int n;
7      std::vector<T> a;
8
9      Fenwick(int n = 0) {
10         init(n);
11     }
12
13     void init(int n) {
14         this->n = n;
15         a.assign(n, T());
16     }
17
18     void add(int x, T v) {
19         for (int i = x + 1; i <= n; i += i & -i) {
20             a[i - 1] += v;
21         }
22     }
23
24     T sum(int x) {
25         auto ans = T();
26         for (int i = x; i > 0; i -= i & -i) {
27             ans += a[i - 1];
28         }
29         return ans;
30     }
31
32     T rangeSum(int l, int r) {
33         return sum(r) - sum(l);
34     }
35
36     int kth(T k) {
37         int x = 0;
38         for (int i = 1 << std::__lg(n); i; i /= 2) {
39             if (x + i <= n && k >= a[x + i - 1]) {
40                 x += i;
41                 k -= a[x - 1];
42             }
43         }
44         return x;
45     }

```

46 };

6.43 01B - 树状数组 (Fenwick 新版)

Listing 74: ds/01B-Fenwick.hpp

```

1  /** 树状数组 (Fenwick 新版)
2  *   2023-12-28: https://codeforces.com/contest/1915/submission/239262801
3  **/
4  template <typename T>
5  struct Fenwick {
6      int n;
7      std::vector<T> a;
8
9      Fenwick(int n_ = 0) {
10         init(n_);
11     }
12
13     void init(int n_) {
14         n = n_;
15         a.assign(n, T{});
16     }
17
18     void add(int x, const T &v) {
19         for (int i = x + 1; i <= n; i += i & -i) {
20             a[i - 1] = a[i - 1] + v;
21         }
22     }
23
24     T sum(int x) {
25         T ans{};
26         for (int i = x; i > 0; i -= i & -i) {
27             ans = ans + a[i - 1];
28         }
29         return ans;
30     }
31
32     T rangeSum(int l, int r) {
33         return sum(r) - sum(l);
34     }
35
36     int select(const T &k) {
37         int x = 0;
38         T cur{};
39         for (int i = 1 << std::lg(n); i; i /= 2) {
40             if (x + i <= n && cur + a[x + i - 1] <= k) {
41                 x += i;
42                 cur = cur + a[x - 1];
43             }
44         }
45         return x;
46     }
47 };

```

6.44 02 - 并查集 (DSU)

Listing 75: ds/02-DSU.hpp

```

1  /** 并查集 (DSU)
2  *   2023-08-04: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=63239142
3  **/
4  struct DSU {
5      std::vector<int> f, siz;
6

```

```

7   DSU() {}
8   DSU(int n) {
9       init(n);
10  }
11
12  void init(int n) {
13      f.resize(n);
14      std::iota(f.begin(), f.end(), 0);
15      siz.assign(n, 1);
16  }
17
18  int find(int x) {
19      while (x != f[x]) {
20          x = f[x] = f[f[x]];
21      }
22      return x;
23  }
24
25  bool same(int x, int y) {
26      return find(x) == find(y);
27  }
28
29  bool merge(int x, int y) {
30      x = find(x);
31      y = find(y);
32      if (x == y) {
33          return false;
34      }
35      siz[x] += siz[y];
36      f[y] = x;
37      return true;
38  }
39
40  int size(int x) {
41      return siz[find(x)];
42  }
43 };

```

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

Listing 76: ds/03A-Segment-Tree.hpp

```

1  /** 线段树 (SegmentTree+Info 区间加+单点修改)
2   *   2023-09-13: https://qoj.ac/submission/178310
3   **/
4  struct SegmentTree {
5      int n;
6      std::vector<int> tag;
7      std::vector<Info> info;
8      SegmentTree(int n_) : n(n_), tag(4 * n), info(4 * n) {}
9
10     void pull(int p) {
11         info[p] = info[2 * p] + info[2 * p + 1];
12     }
13
14     void add(int p, int v) {
15         tag[p] += v;
16         info[p].max += v;
17     }
18
19     void push(int p) {
20         add(2 * p, tag[p]);
21         add(2 * p + 1, tag[p]);
22         tag[p] = 0;

```

```

23     }
24
25     Info query(int p, int l, int r, int x, int y) {
26         if (l >= y || r <= x) {
27             return {};
28         }
29         if (l >= x && r <= y) {
30             return info[p];
31         }
32         int m = (l + r) / 2;
33         push(p);
34         return query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y);
35     }
36
37     Info query(int x, int y) {
38         return query(1, 0, n, x, y);
39     }
40
41     void rangeAdd(int p, int l, int r, int x, int y, int v) {
42         if (l >= y || r <= x) {
43             return;
44         }
45         if (l >= x && r <= y) {
46             return add(p, v);
47         }
48         int m = (l + r) / 2;
49         push(p);
50         rangeAdd(2 * p, l, m, x, y, v);
51         rangeAdd(2 * p + 1, m, r, x, y, v);
52         pull(p);
53     }
54
55     void rangeAdd(int x, int y, int v) {
56         rangeAdd(1, 0, n, x, y, v);
57     }
58
59     void modify(int p, int l, int r, int x, const Info &v) {
60         if (r - l == 1) {
61             info[p] = v;
62             return;
63         }
64         int m = (l + r) / 2;
65         push(p);
66         if (x < m) {
67             modify(2 * p, l, m, x, v);
68         } else {
69             modify(2 * p + 1, m, r, x, v);
70         }
71         pull(p);
72     }
73
74     void modify(int x, const Info &v) {
75         modify(1, 0, n, x, v);
76     }
77 };

```

6.46 03B - 线段树 (SegmentTree 区间乘 + 单点加)

Listing 77: ds/03B-Segment-Tree.hpp

```

1  /** 线段树 (SegmentTree 区间乘+单点加)
2  *   2023-10-18: https://cf.dianhsu.com/gym/104417/submission/223800089
3  **/

```

```

4 struct SegmentTree {
5     int n;
6     std::vector<int> tag, sum;
7     SegmentTree(int n_) : n(n_), tag(4 * n, 1), sum(4 * n) {}
8
9     void pull(int p) {
10         sum[p] = (sum[2 * p] + sum[2 * p + 1]) % P;
11     }
12
13     void mul(int p, int v) {
14         tag[p] = 1LL * tag[p] * v % P;
15         sum[p] = 1LL * sum[p] * v % P;
16     }
17
18     void push(int p) {
19         mul(2 * p, tag[p]);
20         mul(2 * p + 1, tag[p]);
21         tag[p] = 1;
22     }
23
24     int query(int p, int l, int r, int x, int y) {
25         if (l >= y || r <= x) {
26             return 0;
27         }
28         if (l >= x && r <= y) {
29             return sum[p];
30         }
31         int m = (l + r) / 2;
32         push(p);
33         return (query(2 * p, l, m, x, y) + query(2 * p + 1, m, r, x, y)) % P;
34     }
35
36     int query(int x, int y) {
37         return query(1, 0, n, x, y);
38     }
39
40     void rangeMul(int p, int l, int r, int x, int y, int v) {
41         if (l >= y || r <= x) {
42             return;
43         }
44         if (l >= x && r <= y) {
45             return mul(p, v);
46         }
47         int m = (l + r) / 2;
48         push(p);
49         rangeMul(2 * p, l, m, x, y, v);
50         rangeMul(2 * p + 1, m, r, x, y, v);
51         pull(p);
52     }
53
54     void rangeMul(int x, int y, int v) {
55         rangeMul(1, 0, n, x, y, v);
56     }
57
58     void add(int p, int l, int r, int x, int v) {
59         if (r - l == 1) {
60             sum[p] = (sum[p] + v) % P;
61             return;
62         }
63         int m = (l + r) / 2;
64         push(p);
65         if (x < m) {
66             add(2 * p, l, m, x, v);

```

```

67     } else {
68         add(2 * p + 1, m, r, x, v);
69     }
70     pull(p);
71 }
72
73 void add(int x, int v) {
74     add(1, 0, n, x, v);
75 }
76 };

```

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.hpp

```

1  /** 线段树 (SegmentTree+Info 初始赋值+单点修改+查找前驱后继)
2  *   2023-07-17: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=62804432
3  *   2024-06-25: https://codeforces.com/contest/1982/submission/267353839
4  **/
5  template<class Info>
6  struct SegmentTree {
7      int n;
8      std::vector<Info> info;
9      SegmentTree() : n(0) {}
10     SegmentTree(int n_, Info v_ = Info()) {
11         init(n_, v_);
12     }
13     template<class T>
14     SegmentTree(std::vector<T> init_) {
15         init(init_);
16     }
17     void init(int n_, Info v_ = Info()) {
18         init(std::vector(n_, v_));
19     }
20     template<class T>
21     void init(std::vector<T> init_) {
22         n = init_.size();
23         info.assign(4 << std::lg(n), Info());
24         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
25             if (r - l == 1) {
26                 info[p] = init_[l];
27                 return;
28             }
29             int m = (l + r) / 2;
30             build(2 * p, l, m);
31             build(2 * p + 1, m, r);
32             pull(p);
33         };
34         build(1, 0, n);
35     }
36     void pull(int p) {
37         info[p] = info[2 * p] + info[2 * p + 1];
38     }
39     void modify(int p, int l, int r, int x, const Info &v) {
40         if (r - l == 1) {
41             info[p] = v;
42             return;
43         }
44         int m = (l + r) / 2;
45         if (x < m) {
46             modify(2 * p, l, m, x, v);
47         } else {
48             modify(2 * p + 1, m, r, x, v);

```



```

49     }
50     pull(p);
51 }
52 void modify(int p, const Info &v) {
53     modify(1, 0, n, p, v);
54 }
55 Info rangeQuery(int p, int l, int r, int x, int y) {
56     if (l >= y || r <= x) {
57         return Info();
58     }
59     if (l >= x && r <= y) {
60         return info[p];
61     }
62     int m = (l + r) / 2;
63     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
64 }
65 Info rangeQuery(int l, int r) {
66     return rangeQuery(1, 0, n, l, r);
67 }
68 template<class F>
69 int findFirst(int p, int l, int r, int x, int y, F &&pred) {
70     if (l >= y || r <= x) {
71         return -1;
72     }
73     if (l >= x && r <= y && !pred(info[p])) {
74         return -1;
75     }
76     if (r - l == 1) {
77         return l;
78     }
79     int m = (l + r) / 2;
80     int res = findFirst(2 * p, l, m, x, y, pred);
81     if (res == -1) {
82         res = findFirst(2 * p + 1, m, r, x, y, pred);
83     }
84     return res;
85 }
86 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.hpp

```

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;
33         if (x < m) {
34             modify(2 * p, l, m, x, v);
35         } else {
36             modify(2 * p + 1, m, r, x, v);
37         }
38         pull(p);
39     }
40     void modify(int p, const Info &v) {
41         modify(1, 0, n, p, v);
42     }
43     Info rangeQuery(int p, int l, int r, int x, int y) {
44         if (l >= y || r <= x) {
45             return Info();
46         }
47         if (l >= x && r <= y) {
48             return info[p];
49         }
50         int m = (l + r) / 2;
51         return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
52     }
53     Info rangeQuery(int l, int r) {
54         return rangeQuery(1, 0, n, l, r);
55     }
56 };

```

6.49 04 - 懒标记线段树 (LazySegmentTree)

Listing 80: ds/04-Lazy-Segt.hpp

```

1  /** 懒标记线段树 (LazySegmentTree)
2   *   2023-03-03: https://atcoder.jp/contests/joi2023yo2/submissions/39363123
3   *   2023-03-12: https://codeforces.com/contest/1804/submission/197106837
4   *   2023-07-17: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=62804432
5   *   2023-11-12: https://qoj.ac/submission/249505
6  **/
7  template<class Info, class Tag>
8  struct LazySegmentTree {
9      int n;
10     std::vector<Info> info;
11     std::vector<Tag> tag;
12     LazySegmentTree() : n(0) {}
13     LazySegmentTree(int n_, Info v_ = Info()) {
14         init(n_, v_);
15     }
16     template<class T>
17     LazySegmentTree(std::vector<T> init_) {
18         init(init_);
19     }
20     void init(int n_, Info v_ = Info()) {
21         init(std::vector(n_, v_));
22     }
23     template<class T>
24     void init(std::vector<T> init_) {
25         n = init_.size();
26         info.assign(4 << std::__lg(n), Info());
27         tag.assign(4 << std::__lg(n), Tag());
28         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
29             if (r - l == 1) {
30                 info[p] = init_[l];
31                 return;
32             }
33             int m = (l + r) / 2;
34             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.hpp

```

1  constexpr int P = 998244353;
2  using i64 = long long;
3  // assume -P <= x < 2P
4  int norm(int x) {
5      if (x < 0) {
6          x += P;
7      }
8      if (x >= P) {
9          x -= P;
10     }
11     return x;
12 }
13 template<class T>
14 T power(T a, i64 b) {
15     T res = 1;
16     for (; b; b /= 2, a *= a) {
17         if (b % 2) {
18             res *= a;
19         }
20     }
21     return res;
22 }
23 struct Z {
24     int x;
25     Z(int x = 0) : x(norm(x)) {}
26     Z(i64 x) : x(norm(x % P)) {}
27     int val() const {

```

```

28     return x;
29 }
30 Z operator-() const {
31     return Z(norm(P - x));
32 }
33 Z inv() const {
34     assert(x != 0);
35     return power(*this, P - 2);
36 }
37 Z &operator*=(const Z &rhs) {
38     x = i64(x) * rhs.x % P;
39     return *this;
40 }
41 Z &operator+=(const Z &rhs) {
42     x = norm(x + rhs.x);
43     return *this;
44 }
45 Z &operator-=(const Z &rhs) {
46     x = norm(x - rhs.x);
47     return *this;
48 }
49 Z &operator/=(const Z &rhs) {
50     return *this *= rhs.inv();
51 }
52 friend Z operator*(const Z &lhs, const Z &rhs) {
53     Z res = lhs;
54     res *= rhs;
55     return res;
56 }
57 friend Z operator+(const Z &lhs, const Z &rhs) {
58     Z res = lhs;
59     res += rhs;
60     return res;
61 }
62 friend Z operator-(const Z &lhs, const Z &rhs) {
63     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.hpp

```

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) & {
170         return *this *= rhs.inv();
171     }
172     friend constexpr MInt operator*(MInt lhs, MInt rhs) {
173         MInt res = lhs;
174         res *= rhs;
175         return res;
176     }
177     friend constexpr MInt operator+(MInt lhs, MInt rhs) {
178         MInt res = lhs;
179         res += rhs;
180         return res;
181     }
182     friend constexpr MInt operator-(MInt lhs, MInt rhs) {
183         MInt res = lhs;
184         res -= rhs;
185         return res;
186     }
187     friend constexpr MInt operator/(MInt lhs, MInt rhs) {
188         MInt res = lhs;
189         res /= rhs;
190         return res;
191     }
192     friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
193         i64 v;
194         is >> v;
195         a = MInt(v);
196         return is;

```

```

197     }
198     friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a) {
199         return os << a.val();
200     }
201     friend constexpr bool operator==(MInt lhs, MInt rhs) {
202         return lhs.val() == rhs.val();
203     }
204     friend constexpr bool operator!=(MInt lhs, MInt rhs) {
205         return lhs.val() != rhs.val();
206     }
207 };
208
209 template <>
210 int MInt<0>::Mod = 998244353;
211
212 template <int V, int P>
213 constexpr MInt<P> CInv = MInt<P>(V).inv();
214
215 constexpr int P = 1000000007;
216 using Z = MInt<P>;

```

6.52 05C - 动态取模类 (ModIntBase)

Listing 83: ds/05C-Dynamic-ModInt.hpp

```

1  /** 动态取模类 (ModIntBase)
2   *   2024-08-14: https://ac.nowcoder.com/acm/contest/view-submission?submissionId=70980889&returnHomeType=1&uid=329687984
3   */
4  // TODO: Dynamic ModInt
5
6  template <typename T>
7  constexpr T power(T a, u64 b) {
8      T res{1};
9      for (; b != 0; b /= 2, a *= a) {
10         if (b % 2 == 1) {
11             res *= a;
12         }
13     }
14     return res;
15 }
16
17 template <u32 P>
18 constexpr u32 mulMod(u32 a, u32 b) {
19     return 1ULL * a * b % P;
20 }
21
22 template <u64 P>
23 constexpr u64 mulMod(u64 a, u64 b) {
24     u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
25     res %= P;
26     return res;
27 }
28
29 template <typename U, U P>
30     requires std::unsigned_integral<U>
31 struct ModIntBase {
32 public:
33     constexpr ModIntBase() : x{0} {}
34
35     template <typename T>
36         requires std::integral<T>
37     constexpr ModIntBase(T x_) : x{norm(x_ % T{P})} {}
38

```

```

39     constexpr static U norm(U x) {
40         if ((x >> (8 * sizeof(U) - 1) & 1) == 1) {
41             x += P;
42         }
43         if (x >= P) {
44             x -= P;
45         }
46         return x;
47     }
48
49     constexpr U val() const {
50         return x;
51     }
52
53     constexpr ModIntBase operator-() const {
54         ModIntBase res;
55         res.x = norm(P - x);
56         return res;
57     }
58
59     constexpr ModIntBase inv() const {
60         return power(*this, P - 2);
61     }
62
63     constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
64         x = mulMod<P>(x, rhs.val());
65         return *this;
66     }
67
68     constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
69         x = norm(x + rhs.x);
70         return *this;
71     }
72
73     constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
74         x = norm(x - rhs.x);
75         return *this;
76     }
77
78     constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
79         return *this *= rhs.inv();
80     }
81
82     friend constexpr ModIntBase operator*(ModIntBase lhs, const ModIntBase &rhs) {
83         lhs *= rhs;
84         return lhs;
85     }
86
87     friend constexpr ModIntBase operator+(ModIntBase lhs, const ModIntBase &rhs) {
88         lhs += rhs;
89         return lhs;
90     }
91
92     friend constexpr ModIntBase operator-(ModIntBase lhs, const ModIntBase &rhs) {
93         lhs -= rhs;
94         return lhs;
95     }
96
97     friend constexpr ModIntBase operator/(ModIntBase lhs, const ModIntBase &rhs) {
98         lhs /= rhs;
99         return lhs;
100    }
101
102    friend constexpr std::ostream &operator<<(std::ostream &os, const ModIntBase &a) {

```

```

103     return os << a.val();
104 }
105
106 friend constexpr bool operator==(ModIntBase lhs, ModIntBase rhs) {
107     return lhs.val() == rhs.val();
108 }
109
110 friend constexpr bool operator!=(ModIntBase lhs, ModIntBase rhs) {
111     return lhs.val() != rhs.val();
112 }
113
114 friend constexpr bool operator<(ModIntBase lhs, ModIntBase rhs) {
115     return lhs.val() < rhs.val();
116 }
117
118 private:
119     U x;
120 };
121
122 template <u32 P>
123 using ModInt = ModIntBase<u32, P>;
124
125 template <u64 P>
126 using ModInt64 = ModIntBase<u64, P>;
127
128 constexpr u32 P = 998244353;
129 using Z = ModInt<P>;

```

6.53 06 - 状压 RMQ (RMQ)

Listing 84: ds/06-RMQ.hpp

```

1  /** 状压RMQ (RMQ)
2   *   2023-03-02: https://atcoder.jp/contests/joi2022ho/submissions/39351739
3   *   2023-09-04: https://qoj.ac/submission/163598
4   **/
5  template<class T,
6      class Cmp = std::less<T>>
7  struct RMQ {
8      const Cmp cmp = Cmp();
9      static constexpr unsigned B = 64;
10     using u64 = unsigned long long;
11     int n;
12     std::vector<std::vector<T>>> a;
13     std::vector<T> pre, suf, ini;
14     std::vector<u64> stk;
15     RMQ() {}
16     RMQ(const std::vector<T> &v) {
17         init(v);
18     }
19     void init(const std::vector<T> &v) {
20         n = v.size();
21         pre = suf = ini = v;
22         stk.resize(n);
23         if (!n) {
24             return;
25         }
26         const int M = (n - 1) / B + 1;
27         const int lg = std::__lg(M);
28         a.assign(lg + 1, std::vector<T>(M));
29         for (int i = 0; i < M; i++) {
30             a[0][i] = v[i * B];
31             for (int j = 1; j < B && i * B + j < n; j++) {
32                 a[0][i] = std::min(a[0][i], v[i * B + j], cmp);

```

```

33     }
34 }
35 for (int i = 1; i < n; i++) {
36     if (i % B) {
37         pre[i] = std::min(pre[i], pre[i - 1], cmp);
38     }
39 }
40 for (int i = n - 2; i >= 0; i--) {
41     if (i % B != B - 1) {
42         suf[i] = std::min(suf[i], suf[i + 1], cmp);
43     }
44 }
45 for (int j = 0; j < lg; j++) {
46     for (int i = 0; i + (2 << j) <= M; i++) {
47         a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
48     }
49 }
50 for (int i = 0; i < M; i++) {
51     const int l = i * B;
52     const int r = std::min(1U * n, l + B);
53     u64 s = 0;
54     for (int j = l; j < r; j++) {
55         while (s && cmp(v[j], v[std::__lg(s) + l])) {
56             s ^= 1ULL << std::__lg(s);
57         }
58         s |= 1ULL << (j - l);
59         stk[j] = s;
60     }
61 }
62 }
63 T operator()(int l, int r) {
64     if (l / B != (r - 1) / B) {
65         T ans = std::min(suf[l], pre[r - 1], cmp);
66         l = l / B + 1;
67         r = r / B;
68         if (l < r) {
69             int k = std::__lg(r - l);
70             ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
71         }
72         return ans;
73     } else {
74         int x = B * (l / B);
75         return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
76     }
77 }
78 };

```

6.54 07A - Splay

Listing 85: ds/07A-Splay.hpp

```

1 struct Node {
2     Node *l = nullptr;
3     Node *r = nullptr;
4     int cnt = 0;
5     i64 sum = 0;
6 };
7
8 Node *add(Node *t, int l, int r, int p, int v) {
9     Node *x = new Node;
10    if (t) {
11        *x = *t;
12    }

```

```

13     x->cnt += 1;
14     x->sum += v;
15     if (r - l == 1) {
16         return x;
17     }
18     int m = (l + r) / 2;
19     if (p < m) {
20         x->l = add(x->l, l, m, p, v);
21     } else {
22         x->r = add(x->r, m, r, p, v);
23     }
24     return x;
25 }
26
27 int find(Node *tl, Node *tr, int l, int r, int x) {
28     if (r <= x) {
29         return -1;
30     }
31     if (l >= x) {
32         int cnt = (tr ? tr->cnt : 0) - (tl ? tl->cnt : 0);
33         if (cnt == 0) {
34             return -1;
35         }
36         if (r - l == 1) {
37             return l;
38         }
39     }
40     int m = (l + r) / 2;
41     int res = find(tl->l : tl, tr ? tr->l : tr, l, m, x);
42     if (res == -1) {
43         res = find(tl ? tl->r : tl, tr ? tr->r : tr, m, r, x);
44     }
45     return res;
46 }
47
48 std::pair<int, i64> get(Node *t, int l, int r, int x, int y) {
49     if (l >= y || r <= x || !t) {
50         return {0, 0LL};
51     }
52     if (l >= x && r <= y) {
53         return {t->cnt, t->sum};
54     }
55     int m = (l + r) / 2;
56     auto [cl, sl] = get(t->l, l, m, x, y);
57     auto [cr, sr] = get(t->r, m, r, x, y);
58     return {cl + cr, sl + sr};
59 }
60
61 struct Tree {
62     int add = 0;
63     int val = 0;
64     int id = 0;
65     Tree *ch[2] = {};
66     Tree *p = nullptr;
67 };
68
69 int pos(Tree *t) {
70     return t->p->ch[1] == t;
71 }
72
73 void add(Tree *t, int v) {
74     t->val += v;
75     t->add += v;
76 }

```

```

77
78 void push(Tree *t) {
79     if (t->ch[0]) {
80         add(t->ch[0], t->add);
81     }
82     if (t->ch[1]) {
83         add(t->ch[1], t->add);
84     }
85     t->add = 0;
86 }
87
88 void rotate(Tree *t) {
89     Tree *q = t->p;
90     int x = !pos(t);
91     q->ch[!x] = t->ch[x];
92     if (t->ch[x]) t->ch[x]->p = q;
93     t->p = q->p;
94     if (q->p) q->p->ch[pos(q)] = t;
95     t->ch[x] = q;
96     q->p = t;
97 }
98
99 void splay(Tree *t) {
100     std::vector<Tree *> s;
101     for (Tree *i = t; i->p; i = i->p) s.push_back(i->p);
102     while (!s.empty()) {
103         push(s.back());
104         s.pop_back();
105     }
106     push(t);
107     while (t->p) {
108         if (t->p->p) {
109             if (pos(t) == pos(t->p)) rotate(t->p);
110             else rotate(t);
111         }
112         rotate(t);
113     }
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) {
178     if (!l) {
179         return r;
180     }
181     if (!r) {
182         return l;
183     }
184     Tree *i = l;
185     while (i->ch[1]) {
186         i = i->ch[1];
187     }
188     splay(i);
189     i->ch[1] = r;
190     r->p = i;
191     return i;
192 }

```

6.55 07B - Splay

Listing 86: ds/07B-Splay.hpp

```

1 struct Node {
2     Node *ch[2], *p;
3     bool rev;
4     int siz = 1;
5     Node() : ch{nullptr, nullptr}, p(nullptr), rev(false) {}
6 };

```

```

7 void reverse(Node *t) {
8     if (t) {
9         std::swap(t->ch[0], t->ch[1]);
10        t->rev ^= 1;
11    }
12 }
13 void push(Node *t) {
14     if (t->rev) {
15         reverse(t->ch[0]);
16         reverse(t->ch[1]);
17         t->rev = false;
18     }
19 }
20 void pull(Node *t) {
21     t->siz = (t->ch[0] ? t->ch[0]->siz : 0) + 1 + (t->ch[1] ? t->ch[1]->siz : 0);
22 }
23 bool isroot(Node *t) {
24     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
25 }
26 int pos(Node *t) {
27     return t->p->ch[1] == t;
28 }
29 void pushAll(Node *t) {
30     if (!isroot(t)) {
31         pushAll(t->p);
32     }
33     push(t);
34 }
35 void rotate(Node *t) {
36     Node *q = t->p;
37     int x = !pos(t);
38     q->ch[!x] = t->ch[x];
39     if (t->ch[x]) {
40         t->ch[x]->p = q;
41     }
42     t->p = q->p;
43     if (!isroot(q)) {
44         q->p->ch[pos(q)] = t;
45     }
46     t->ch[x] = q;
47     q->p = t;
48     pull(q);
49 }
50 void splay(Node *t) {
51     pushAll(t);
52     while (!isroot(t)) {
53         if (!isroot(t->p)) {
54             if (pos(t) == pos(t->p)) {
55                 rotate(t->p);
56             } else {
57                 rotate(t);
58             }
59         }
60         rotate(t);
61     }
62     pull(t);
63 }
64 void access(Node *t) {
65     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
66         splay(i);
67         i->ch[1] = q;
68         pull(i);
69     }

```

```

70     splay(t);
71 }
72 void makeroot(Node *t) {
73     access(t);
74     reverse(t);
75 }
76 void link(Node *x, Node *y) {
77     makeroot(x);
78     x->p = y;
79 }
80 void split(Node *x, Node *y) {
81     makeroot(x);
82     access(y);
83 }
84 void cut(Node *x, Node *y) {
85     split(x, y);
86     x->p = y->ch[0] = nullptr;
87     pull(y);
88 }
89 int dist(Node *x, Node *y) {
90     split(x, y);
91     return y->siz - 1;
92 }

```

6.56 07C - Splay

Listing 87: ds/07C-Splay.hpp

```

1  struct Matrix : std::array<std::array<i64, 4>, 4> {
2      Matrix(i64 v = 0) {
3          for (int i = 0; i < 4; i++) {
4              for (int j = 0; j < 4; j++) {
5                  (*this)[i][j] = (i == j ? v : inf);
6              }
7          }
8      }
9  };
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     }
74     t->ch[x] = q;
75     q->p = t;
76     pull(q);
77 }
78 void splay(Node *t) {
79     pushAll(t);
80     while (!isroot(t)) {
81         if (!isroot(t->p)) {
82             if (pos(t) == pos(t->p)) {
83                 rotate(t->p);
84             } else {
85                 rotate(t);
86             }
87         }
88         rotate(t);
89     }
90     pull(t);
91 }
92
93 std::array<i64, 4> get(Node *t) {
94     std::array<i64, 4> ans;
95     ans.fill(inf);
96     ans[3] = 0;
97     for (int i = 0; i < 3; i++) {
98         for (int j = 0; j < 4; j++) {

```

```

99         ans[i] = std::min(ans[i], t->prd[i][j]);
100     }
101 }
102 return ans;
103 }
104
105 void access(Node *t) {
106     std::array<i64, 4> old{};
107     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
108         splay(i);
109         if (i->ch[1]) {
110             auto res = get(i->ch[1]);
111             i->sumg += res[0];
112             i->sumh += std::min({res[1], res[2], res[3]});
113             i->sumb += std::min({res[0], res[1], res[2], res[3]});
114         }
115         i->ch[1] = q;
116         i->sumg -= old[0];
117         i->sumh -= std::min({old[1], old[2], old[3]});
118         i->sumb -= std::min({old[0], old[1], old[2], old[3]});
119         old = get(i);
120         i->update();
121         pull(i);
122     }
123     splay(t);
124 }
125 ...
126
127 ```cpp
128 /** Splay
129  * 2024-06-24: https://cf.dianhsu.com/gym/105229/submission/267199687?version=1.5
130  */
131 constexpr int D = 27;
132 struct Info {
133     int up[D][2]{};
134     int down[D][2]{};
135     int t = 0;
136     i64 ans = 0;
137 };
138
139 Info operator+(const Info &a, const Info &b) {
140     Info c;
141     c.t = a.t ^ b.t;
142     c.ans = a.ans + b.ans;
143     for (int i = 0; i < D; i++) {
144         for (int j = 0; j < 2; j++) {
145             c.ans += (1LL << i) * a.down[i][j] * b.up[i][j ^ 1];
146             c.up[i][j] += a.up[i][j] + b.up[i][j ^ (a.t >> i & 1)];
147             c.down[i][j] += b.down[i][j] + a.down[i][j ^ (b.t >> i & 1)];
148         }
149     }
150     return c;
151 }
152 struct Node {
153     Node *ch[2], *p;
154     Info val;
155     Info tot;
156     int cnt[D][2];
157     i64 pair[D][2];
158     i64 sum;
159     Node() : ch{nullptr, nullptr}, p(nullptr), cnt{}, pair{}, sum{} {}
160 };
161 void pull(Node *t) {
162     t->tot = (t->ch[0] ? t->ch[0]->tot : Info{}) + t->val + (t->ch[1] ? t->ch[1]->tot : Info{});

```

```

163 }
164 bool isroot(Node *t) {
165     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
166 }
167 int pos(Node *t) {
168     return t->p->ch[1] == t;
169 }
170 void rotate(Node *t) {
171     Node *q = t->p;
172     int x = !pos(t);
173     q->ch[!x] = t->ch[x];
174     if (t->ch[x]) {
175         t->ch[x]->p = q;
176     }
177     t->p = q->p;
178     if (!isroot(q)) {
179         q->p->ch[pos(q)] = t;
180     }
181     t->ch[x] = q;
182     q->p = t;
183     pull(q);
184 }
185 void update(Node *t) {
186     t->val.ans = t->val.t + t->sum;
187     for (int i = 0; i < D; i++) {
188         t->val.ans += (1LL << i) * t->pair[i][t->val.t >> i & 1];
189         for (int j = 0; j < 2; j++) {
190             t->val.up[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
191             t->val.down[i][j] = t->cnt[i][j ^ (t->val.t >> i & 1)];
192         }
193         t->val.up[i][t->val.t >> i & 1]++;
194         t->val.down[i][t->val.t >> i & 1]++;
195     }
196     pull(t);
197 }
198 void splay(Node *t) {
199     while (!isroot(t)) {
200         if (!isroot(t->p)) {
201             if (pos(t) == pos(t->p)) {
202                 rotate(t->p);
203             } else {
204                 rotate(t);
205             }
206         }
207         rotate(t);
208     }
209     pull(t);
210 }
211 void add(Node *t, Info s) {
212     for (int i = 0; i < D; i++) {
213         for (int x = 0; x < 2; x++) {
214             t->pair[i][x] += s.up[i][1 ^ x];
215             for (int j = 0; j < 2; j++) {
216                 t->pair[i][x] += t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
217             }
218         }
219         for (int j = 0; j < 2; j++) {
220             t->cnt[i][j] += s.up[i][j];
221         }
222     }
223     t->sum += s.ans;
224 }
225 void del(Node *t, Info s) {

```

```

226     t->sum -= s.ans;
227     for (int i = 0; i < D; i++) {
228         for (int j = 0; j < 2; j++) {
229             t->cnt[i][j] -= s.up[i][j];
230         }
231         for (int x = 0; x < 2; x++) {
232             for (int j = 0; j < 2; j++) {
233                 t->pair[i][x] -= t->cnt[i][j] * s.up[i][j ^ 1 ^ x];
234             }
235             t->pair[i][x] -= s.up[i][1 ^ x];
236         }
237     }
238 }
239 void access(Node *t, int v) {
240     Info lst;
241     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
242         splay(i);
243         if (i->ch[1]) {
244             add(i, i->ch[1]->tot);
245         }
246         i->ch[1] = q;
247         if (q) {
248             del(i, lst);
249         } else {
250             i->val.t = v;
251         }
252         lst = i->tot;
253         update(i);
254     }
255     splay(t);
256 }

```

6.57 08A - 其他平衡树

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

```

1  struct Node {
2      Node *l = nullptr;
3      Node *r = nullptr;
4      int sum = 0;
5      int sumodd = 0;
6
7      Node(Node *t) {
8          if (t) {
9              *this = *t;
10         }
11     }
12 };
13
14 Node *add(Node *t, int l, int r, int x, int v) {
15     t = new Node(t);
16     t->sum += v;
17     t->sumodd += (x % 2) * v;
18     if (r - l == 1) {
19         return t;
20     }
21     int m = (l + r) / 2;
22     if (x < m) {
23         t->l = add(t->l, l, m, x, v);
24     } else {
25         t->r = add(t->r, m, r, x, v);
26     }
27     return t;

```

```

28 }
29
30 int query1(Node *t1, Node *t2, int l, int r, int k) {
31     if (r - l == 1) {
32         return l;
33     }
34     int m = (l + r) / 2;
35     int odd = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
36     int cnt = (t1 && t1->r ? t1->r->sum : 0) - (t2 && t2->r ? t2->r->sum : 0);
37     if (odd > 0 || cnt > k) {
38         return query1(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
39     } else {
40         return query1(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
41     }
42 }
43
44 std::array<int, 3> query2(Node *t1, Node *t2, int l, int r, int k) {
45     if (r - l == 1) {
46         int cnt = (t1 ? t1->sumodd : 0) - (t2 ? t2->sumodd : 0);
47         return {l, cnt, k};
48     }
49     int m = (l + r) / 2;
50     int cnt = (t1 && t1->r ? t1->r->sumodd : 0) - (t2 && t2->r ? t2->r->sumodd : 0);
51     if (cnt > k) {
52         return query2(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, k);
53     } else {
54         return query2(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, k - cnt);
55     }
56 }

```

6.58 08B - 其他平衡树

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

```

1 struct Node {
2     Node *l = nullptr;
3     Node *r = nullptr;
4     int cnt = 0;
5 };
6
7 Node *add(Node *t, int l, int r, int x) {
8     if (t) {
9         t = new Node(*t);
10    } else {
11        t = new Node;
12    }
13    t->cnt += 1;
14    if (r - l == 1) {
15        return t;
16    }
17    int m = (l + r) / 2;
18    if (x < m) {
19        t->l = add(t->l, l, m, x);
20    } else {
21        t->r = add(t->r, m, r, x);
22    }
23    return t;
24 }
25
26 int query(Node *t1, Node *t2, int l, int r, int x) {
27     int cnt = (t2 ? t2->cnt : 0) - (t1 ? t1->cnt : 0);
28     if (cnt == 0 || l >= x) {
29         return -1;

```



```

30     }
31     if (r - l == 1) {
32         return l;
33     }
34     int m = (l + r) / 2;
35     int res = query(t1 ? t1->r : t1, t2 ? t2->r : t2, m, r, x);
36     if (res == -1) {
37         res = query(t1 ? t1->l : t1, t2 ? t2->l : t2, l, m, x);
38     }
39     return res;
40 }

```

6.59 08C - 其他平衡树

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

```

1  struct Info {
2      int imp = 0;
3      int id = 0;
4  };
5
6  Info operator+(Info a, Info b) {
7      return {std::max(a.imp, b.imp), 0};
8  }
9
10 struct Node {
11     int w = rng();
12     Info info;
13     Info sum;
14     int siz = 1;
15     Node *l = nullptr;
16     Node *r = nullptr;
17 };
18
19 void pull(Node *t) {
20     t->sum = t->info;
21     t->siz = 1;
22     if (t->l) {
23         t->sum = t->l->sum + t->sum;
24         t->siz += t->l->siz;
25     }
26     if (t->r) {
27         t->sum = t->sum + t->r->sum;
28         t->siz += t->r->siz;
29     }
30 }
31
32 std::pair<Node *, Node *> splitAt(Node *t, int p) {
33     if (!t) {
34         return {t, t};
35     }
36     if (p <= (t->l ? t->l->siz : 0)) {
37         auto [l, r] = splitAt(t->l, p);
38         t->l = r;
39         pull(t);
40         return {l, t};
41     } else {
42         auto [l, r] = splitAt(t->r, p - 1 - (t->l ? t->l->siz : 0));
43         t->r = l;
44         pull(t);
45         return {t, r};
46     }
47 }
48

```

```

49 void insertAt(Node *t, int p, Node *x) {
50     if (!t) {
51         t = x;
52         return;
53     }
54     if (x->w < t->w) {
55         auto [l, r] = splitAt(t, p);
56         t = x;
57         t->l = l;
58         t->r = r;
59         pull(t);
60         return;
61     }
62     if (p <= (t->l ? t->l->siz : 0)) {
63         insertAt(t->l, p, x);
64     } else {
65         insertAt(t->r, p - 1 - (t->l ? t->l->siz : 0), x);
66     }
67     pull(t);
68 }
69
70 Node *merge(Node *a, Node *b) {
71     if (!a) {
72         return b;
73     }
74     if (!b) {
75         return a;
76     }
77
78     if (a->w < b->w) {
79         a->r = merge(a->r, b);
80         pull(a);
81         return a;
82     } else {
83         b->l = merge(a, b->l);
84         pull(b);
85         return b;
86     }
87 }
88
89 int query(Node *t, int v) {
90     if (!t) {
91         return 0;
92     }
93     if (t->sum.imp < v) {
94         return t->siz;
95     }
96     int res = query(t->r, v);
97     if (res != (t->r ? t->r->siz : 0)) {
98         return res;
99     }
100     if (t->info.imp > v) {
101         return res;
102     }
103     return res + 1 + query(t->l, v);
104 }
105
106 void dfs(Node *t) {
107     if (!t) {
108         return;
109     }
110     dfs(t->l);
111     std::cout << t->info.id << " ";
112     dfs(t->r);

```

113 }

6.60 08D - 其他平衡树

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

```

1 struct Node {
2     Node *l = nullptr;
3     Node *r = nullptr;
4     int cnt = 0;
5     int cntnew = 0;
6 };
7
8 Node *add(int l, int r, int x, int isnew) {
9     Node *t = new Node;
10    t->cnt = 1;
11    t->cntnew = isnew;
12    if (r - l == 1) {
13        return t;
14    }
15    int m = (l + r) / 2;
16    if (x < m) {
17        t->l = add(l, m, x, isnew);
18    } else {
19        t->r = add(m, r, x, isnew);
20    }
21    return t;
22 }
23
24 struct Info {
25     Node *t = nullptr;
26     int psum = 0;
27     bool rev = false;
28 };
29
30 void pull(Node *t) {
31     t->cnt = (t->l ? t->l->cnt : 0) + (t->r ? t->r->cnt : 0);
32     t->cntnew = (t->l ? t->l->cntnew : 0) + (t->r ? t->r->cntnew : 0);
33 }
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.hpp

```

1  template<class T>
2  struct Frac {
3      T num;
4      T den;
5      Frac(T num_, T den_) : num(num_), den(den_) {
6          if (den < 0) {
7              den = -den;
8              num = -num;
9          }
10     }
11     Frac() : Frac(0, 1) {}
12     Frac(T num_) : Frac(num_, 1) {}
13     explicit operator double() const {
14         return 1. * num / den;
15     }
16     Frac &operator+=(const Frac &rhs) {
17         num = num * rhs.den + rhs.num * den;
18         den *= rhs.den;
19         return *this;
20     }
21     Frac &operator-=(const Frac &rhs) {
22         num = num * rhs.den - rhs.num * den;
23         den *= rhs.den;

```

```

24     return *this;
25 }
26 Frac &operator*=(const Frac &rhs) {
27     num *= rhs.num;
28     den *= rhs.den;
29     return *this;
30 }
31 Frac &operator/=(const Frac &rhs) {
32     num *= rhs.den;
33     den *= rhs.num;
34     if (den < 0) {
35         num = -num;
36         den = -den;
37     }
38     return *this;
39 }
40 friend Frac operator+(Frac lhs, const Frac &rhs) {
41     return lhs += rhs;
42 }
43 friend Frac operator-(Frac lhs, const Frac &rhs) {
44     return lhs -= rhs;
45 }
46 friend Frac operator*(Frac lhs, const Frac &rhs) {
47     return lhs *= rhs;
48 }
49 friend Frac operator/(Frac lhs, const Frac &rhs) {
50     return lhs /= rhs;
51 }
52 friend Frac operator~(const Frac &a) {
53     return Frac(-a.num, a.den);
54 }
55 friend bool operator==(const Frac &lhs, const Frac &rhs) {
56     return lhs.num * rhs.den == rhs.num * lhs.den;
57 }
58 friend bool operator!=(const Frac &lhs, const Frac &rhs) {
59     return lhs.num * rhs.den != rhs.num * lhs.den;
60 }
61 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.hpp

```

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.hpp

```

1  /** 高精度 (BigInt)
2   *   2023-09-11: https://qoj.ac/submission/176420
3   **/
4  constexpr int N = 1000;
5
6  struct BigInt {
7      int a[N];
8      BigInt(int x = 0) : a{} {
9          for (int i = 0; x; i++) {
10             a[i] = x % 10;
11             x /= 10;
12         }
13     }
14     BigInt &operator*=(int x) {
15         for (int i = 0; i < N; i++) {
16             a[i] *= x;
17         }
18         for (int i = 0; i < N - 1; i++) {
19             a[i + 1] += a[i] / 10;
20             a[i] %= 10;
21         }
22         return *this;
23     }
24     BigInt &operator/=(int x) {
25         for (int i = N - 1; i >= 0; i--) {
26             if (i) {
27                 a[i - 1] += a[i] % x * 10;
28             }
29             a[i] /= x;
30         }
31         return *this;

```

```

32     }
33     BigInt &operator+=(const BigInt &x) {
34         for (int i = 0; i < N; i++) {
35             a[i] += x.a[i];
36             if (a[i] >= 10) {
37                 a[i + 1] += 1;
38                 a[i] -= 10;
39             }
40         }
41         return *this;
42     }
43 };
44
45 std::ostream &operator<<(std::ostream &o, const BigInt &a) {
46     int t = N - 1;
47     while (a.a[t] == 0) {
48         t--;
49     }
50     for (int i = t; i >= 0; i--) {
51         o << a.a[i];
52     }
53     return o;
54 }

```

7 Watashi 代码库 (备用)

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

Listing 95: **rmq.hpp**

```

1  #include <algorithm> // copy
2  #include <climits>   // CHAR_BIT
3
4  using namespace std;
5
6  template <typename T>
7  struct RMQ {
8      int n;
9      vector<T> e;
10     vector<vector<int>>> rmq;
11
12     static const int INT_BIT = sizeof(4) * CHAR_BIT;
13     static inline int LG2(int i) { return INT_BIT - 1 - __builtin_clz(i); }
14     static inline int BIN(int i) { return 1 << i; }
15
16     int cmp(int l, int r) const {
17         return e[l] <= e[r] ? l : r;
18     }
19
20     void init(int n, const T e[]) {
21         this->n = n;
22         vector<T>(e, e + n).swap(this->e);
23
24         int m = 1;
25         while (BIN(m) <= n) {
26             ++m;
27         }
28         vector<vector<int>>>(m, vector<int>(n)).swap(rmq);
29
30         for (int i = 0; i < n; ++i) {
31             rmq[0][i] = i;
32         }
33         for (int i = 0; BIN(i + 1) <= n; ++i) {

```

```

34         for (int j = 0; j + BIN(i + 1) <= n; ++j) {
35             rmq[i + 1][j] = cmp(rmq[i][j], rmq[i][j + BIN(i)]);
36         }
37     }
38 }
39
40 int index(int l, int r) const {
41     int b = LG2(r - l);
42     return cmp(rmq[b][l], rmq[b][r - (1 << b)]);
43 }
44
45 T value(int l, int r) const {
46     return e[index(l, r)];
47 }
48 };

```

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

Listing 96: lca.hpp

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <vector>
4
5  using namespace std;
6
7  const int MAXM = 16;
8  const int MAXN = 1 << MAXM;
9
10 // LCA
11 struct LCA {
12     vector<int> e[MAXN];
13     int d[MAXN], p[MAXN][MAXM];
14
15     void dfs_(int v, int f) {
16         p[v][0] = f;
17         for (int i = 1; i < MAXM; ++i) {
18             p[v][i] = p[p[v][i - 1]][i - 1];
19         }
20         for (int i = 0; i < (int)e[v].size(); ++i) {
21             int w = e[v][i];
22             if (w != f) {
23                 d[w] = d[v] + 1;
24                 dfs_(w, v);
25             }
26         }
27     }
28
29     int up_(int v, int m) {
30         for (int i = 0; i < MAXM; ++i) {
31             if (m & (1 << i)) {
32                 v = p[v][i];
33             }
34         }
35         return v;
36     }
37
38     int lca(int a, int b) {
39         if (d[a] > d[b]) {
40             swap(a, b);
41         }
42         b = up_(b, d[b] - d[a]);
43         if (a == b) {
44             return a;

```



```

45     } else {
46         for (int i = MAXM - 1; i >= 0; --i) {
47             if (p[a][i] != p[b][i]) {
48                 a = p[a][i];
49                 b = p[b][i];
50             }
51         }
52         return p[a][0];
53     }
54 }
55
56 void init(int n) {
57     for (int i = 0; i < n; ++i) {
58         e[i].clear();
59     }
60 }
61
62 void add(int a, int b) {
63     e[a].push_back(b);
64     e[b].push_back(a);
65 }
66
67 void build() {
68     d[0] = 0;
69     dfs_(0, 0);
70 }
71 } lca;

```

7.3 树状数组

Listing 97: **bit.hpp**

```

1  #include <vector>
2
3  using namespace std;
4
5  template<typename T = int>
6  struct BIT {
7      vector<T> a;
8
9      void init(int n) {
10         vector<T>(n + 1).swap(a);
11     }
12
13     void add(int i, T v) {
14         for (int j = i + 1; j < (int)a.size(); j = (j | (j - 1)) + 1) {
15             a[j] += v;
16         }
17     }
18
19     // [0, i)
20     T sum(int i) const {
21         T ret = T();
22         for (int j = i; j > 0; j = j & (j - 1)) {
23             ret += a[j];
24         }
25         return ret;
26     }
27
28     T get(int i) const {
29         return sum(i + 1) - sum(i);
30     }
31
32     void set(int i, T v) {

```

```

33     add(i, v - get(i));
34 }
35 };

```

7.4 并查集

Listing 98: **union-find.hpp**

```

1  #include <vector>
2
3  using namespace std;
4
5  struct DisjointSet {
6      vector<int> p;
7
8      void init(int n) {
9          p.resize(n);
10         for (int i = 0; i < n; ++i) {
11             p[i] = i;
12         }
13     }
14
15     int getp(int i) {
16         return i == p[i] ? i : (p[i] = getp(p[i]));
17     }
18
19     bool setp(int i, int j) {
20         i = getp(i);
21         j = getp(j);
22         p[i] = j;
23         return i != j;
24     }
25 };

```

7.5 轻重权树剖分

Listing 99: **chain-decomp.hpp**

```

1  #include <cstdio>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int MAXM = 16;
8  const int MAXN = 1 << MAXM;
9
10 // Heavy-Light Decomposition
11 struct TreeDecomposition {
12     vector<int> e[MAXN], c[MAXN];
13     int s[MAXN]; // subtree size
14     int p[MAXN]; // parent id
15     int r[MAXN]; // chain root id
16     int t[MAXN]; // timestamp, index used in segtree
17     int ts;
18
19     void dfs_(int v, int f) {
20         p[v] = f;
21         s[v] = 1;
22         for (int i = 0; i < (int)e[v].size(); ++i) {
23             int w = e[v][i];
24             if (w != f) {
25                 dfs_(w, v);
26                 s[v] += s[w];

```

```

27     }
28   }
29 }
30
31 void decomp_(int v, int f, int k) {
32     t[v] = ts++;
33     c[k].push_back(v);
34     r[v] = k;
35
36     int x = 0, y = -1;
37     for (int i = 0; i < (int)e[v].size(); ++i) {
38         int w = e[v][i];
39         if (w != f) {
40             if (s[w] > x) {
41                 x = s[w];
42                 y = w;
43             }
44         }
45     }
46     if (y != -1) {
47         decomp_(y, v, k);
48     }
49
50     for (int i = 0; i < (int)e[v].size(); ++i) {
51         int w = e[v][i];
52         if (w != f && w != y) {
53             decomp_(w, v, w);
54         }
55     }
56 }
57
58 void init(int n) {
59     for (int i = 0; i < n; ++i) {
60         e[i].clear();
61     }
62 }
63
64 void add(int a, int b) {
65     e[a].push_back(b);
66     e[b].push_back(a);
67 }
68
69 void build() { // !!
70     ts = 0;
71     dfs_(0, 0);
72     decomp_(0, 0, 0);
73 }
74 } hld;

```

7.6 强连通分量

Listing 100: **scc.hpp**

```

1  #include <algorithm>
2  #include <stack>
3  #include <vector>
4
5  using namespace std;
6
7  struct SCCTarjan {
8      int n;
9      vector<vector<int>> e;
10
11      vector<int> id;

```

```

12     vector<vector<int>> scc;
13
14     void init(int n) {
15         this->n = n;
16         vector<vector<int>>(n).swap(e);
17         id.resize(n);
18         dfn.resize(n);
19         low.resize(n);
20     }
21
22     void add(int a, int b) {
23         e[a].push_back(b);
24     }
25
26     vector<int> dfn, low;
27     int timestamp;
28     stack<int> s;
29
30     void dfs(int v) {
31         dfn[v] = timestamp++;
32         low[v] = dfn[v];
33         s.push(v);
34         for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
35             if (dfn[*w] == -1) {
36                 dfs(*w);
37                 low[v] = min(low[v], low[*w]);
38             } else if (dfn[*w] != -2) {
39                 low[v] = min(low[v], dfn[*w]);
40             }
41         }
42
43         if (low[v] == dfn[v]) {
44             vector<int> t;
45             do {
46                 int w = s.top();
47                 s.pop();
48                 id[w] = (int)scc.size();
49                 t.push_back(w);
50                 dfn[w] = -2;
51             } while (t.back() != v);
52             scc.push_back(t);
53         }
54     }
55
56     int gao() {
57         scc.clear();
58         stack<int>().swap(s);
59         timestamp = 0;
60
61         fill(dfn.begin(), dfn.end(), -1);
62         for (int i = 0; i < n; ++i) {
63             if (dfn[i] == -1) {
64                 dfs(i);
65             }
66         }
67         return (int)scc.size();
68     }
69 };

```

7.7 双连通分量

Listing 101: **bcc.hpp**

```

1  #include <algorithm>
2  #include <stack>
3  #include <utility>
4  #include <vector>
5
6  using namespace std;
7
8  // TODO: cannot handle duplicate edges
9  struct Tarjan {
10     int n;
11     vector<vector<int>> e;
12
13     vector<int> cut;
14     vector<pair<int, int>> bridge;
15     vector<vector<pair<int, int>>> bcc;
16
17     void init(int n) {
18         this->n = n;
19         e.clear();
20         e.resize(n);
21         dfn.resize(n);
22         low.resize(n);
23     }
24
25     void add(int a, int b) {
26         // assert(find(e[a].begin(), e[a].end(), b) == e[a].end());
27         e[a].push_back(b);
28         e[b].push_back(a);
29     }
30
31     vector<int> dfn, low;
32     int timestamp;
33     stack<pair<int, int>> s;
34
35     void dfs(int v, int p) {
36         int part = p == -1 ? 0 : 1;
37         dfn[v] = low[v] = timestamp++;
38         for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
39             pair<int, int> f = make_pair(min(v, *w), max(v, *w));
40             if (dfn[*w] == -1) {
41                 s.push(f);
42                 dfs(*w, v);
43                 low[v] = min(low[v], low[*w]);
44                 if (dfn[v] <= low[*w]) {
45                     // articulation point
46                     if (++part == 2) {
47                         cut.push_back(v);
48                     }
49                     // articulation edge
50                     if (dfn[v] < low[*w]) {
51                         bridge.push_back(f);
52                     }
53                     // biconnected component (2-vertex-connected)
54                     vector<pair<int, int>> t;
55                     do {
56                         t.push_back(s.top());
57                         s.pop();
58                     } while (t.back() != f);
59                     bcc.push_back(t);
60                 }
61             } else if (*w != p && dfn[*w] < dfn[v]) {
62                 s.push(f);

```

```

63         low[v] = min(low[v], dfn[*w]);
64     }
65 }
66 }
67
68 void gao() {
69     cut.clear();
70     bridge.clear();
71     bcc.clear();
72
73     timestamp = 0;
74     stack<pair<int, int>>().swap(s);
75     fill(dfn.begin(), dfn.end(), -1);
76
77     for (int i = 0; i < n; ++i) {
78         if (dfn[i] == -1) {
79             dfs(i, -1);
80         }
81     }
82 }
83 };
84
85 struct BridgeBlockTree {
86     Tarjan<MAXN> bcc;
87     DisjointSet<MAXN> ds;
88     vector<int> e[MAXN];
89
90     void init(int n) {
91         bcc.init(n);
92         ds.init(n);
93     }
94
95     void add(int a, int b) {
96         bcc.add(a, b);
97     }
98
99     void gao() {
100         bcc.gao();
101         for (const auto &i : bcc.bcc) {
102             if (i.size() > 1) {
103                 for (const auto &j : i) {
104                     ds.setp(j.first, j.second);
105                 }
106             }
107         }
108         for (const auto &i : bcc.bridge) {
109             int a = ds.getp(i.first);
110             int b = ds.getp(i.second);
111             e[a].push_back(b);
112             e[b].push_back(a);
113         }
114     }
115
116     int id(int v) {
117         return ds.getp(v);
118     }
119 };

```

7.8 二分图匹配

Listing 102: **bimatch.hpp**

```

1 // maximum matchings in bipartite graphs

```

```

2 // maximum cardinality bipartite matching
3 //  $O(|V||E|)$ , generally fast
4
5 #include <algorithm>
6 #include <string>
7 #include <vector>
8
9 using namespace std;
10
11 struct Hungarian {
12     int nx, ny;
13     vector<int> mx, my;
14     vector<vector<int>> e;
15
16     void init(int nx, int ny) {
17         this->nx = nx;
18         this->ny = ny;
19         mx.resize(nx);
20         my.resize(ny);
21         e.clear();
22         e.resize(nx);
23         mark.resize(nx);
24     }
25
26     void add(int a, int b) {
27         e[a].push_back(b);
28     }
29
30     // vector<bool> is evil!!!
31     basic_string<bool> mark;
32
33     bool augment(int i) {
34         if (!mark[i]) {
35             mark[i] = true;
36             for (vector<int>::const_iterator j = e[i].begin(); j != e[i].end(); ++j) {
37                 if (my[*j] == -1 || augment(my[*j])) {
38                     mx[i] = *j;
39                     my[*j] = i;
40                     return true;
41                 }
42             }
43         }
44         return false;
45     }
46
47     int gao() {
48         int ret = 0;
49         fill(mx.begin(), mx.end(), -1);
50         fill(my.begin(), my.end(), -1);
51         for (int i = 0; i < nx; ++i) {
52             fill(mark.begin(), mark.end(), false);
53             if (augment(i)) {
54                 ++ret;
55             }
56         }
57         return ret;
58     }
59 };

```

7.9 最小费用最大流

Listing 103: flow.hpp

```

1 #include <algorithm>

```

```

2  #include <cstdio>
3  #include <limits>
4  #include <queue>
5  #include <vector>
6
7  using namespace std;
8
9  template <int MAXN, typename T = int, typename S = T>
10 struct MinCostMaxFlow {
11     struct NegativeCostCircuitExistsException {
12     };
13
14     struct Edge {
15         int v;
16         T c;
17         S w;
18         int b;
19         Edge(int v, T c, S w, int b) : v(v), c(c), w(w), b(b) {}
20     };
21
22     int n, source, sink;
23     vector<Edge> e[MAXN];
24
25     void init(int n, int source, int sink) {
26         this->n = n;
27         this->source = source;
28         this->sink = sink;
29         for (int i = 0; i < n; ++i) {
30             e[i].clear();
31         }
32     }
33
34     void addEdge(int a, int b, T c, S w) {
35         e[a].push_back(Edge(b, c, w, e[b].size()));
36         e[b].push_back(Edge(a, 0, -w, e[a].size() - 1)); // TODO
37     }
38
39     bool mark[MAXN];
40     T maxc[MAXN];
41     S minw[MAXN];
42     int dist[MAXN];
43     Edge *prev[MAXN];
44
45     bool _spfa() {
46         queue<int> q;
47         fill(mark, mark + n, false);
48         fill(maxc, maxc + n, 0);
49         fill(minw, minw + n, numeric_limits<S>::max());
50         fill(dist, dist + n, 0);
51         fill(prev, prev + n, (Edge *)NULL);
52         mark[source] = true;
53         maxc[source] = numeric_limits<S>::max();
54         minw[source] = 0;
55
56         q.push(source);
57         while (!q.empty()) {
58             int cur = q.front();
59             mark[cur] = false;
60             q.pop();
61             for (typename vector<Edge>::iterator it = e[cur].begin(); it != e[cur].end(); ++it) {
62                 T c = min(maxc[cur], it->c);
63                 if (c == 0) {
64                     continue;
65                 }

```



```

66
67     int v = it->v;
68     S w = minw[cur] + it->w;
69     if (minw[v] > w || (minw[v] == w && maxc[v] < c)) { // TODO
70         maxc[v] = c;
71         minw[v] = w;
72         dist[v] = dist[cur] + 1;
73         if (dist[v] >= n) {
74             return false;
75         }
76         prev[v] = &*it;
77         if (!mark[v]) {
78             mark[v] = true;
79             q.push(v);
80         }
81     }
82 }
83 }
84 return true;
85 }
86
87 pair<T, S> gao() {
88     T sumc = 0;
89     S sumw = 0;
90     while (true) {
91         if (!_spfa()) {
92             throw NegativeCostCircuitExistsException();
93         } else if (maxc[sink] == 0) {
94             break;
95         } else {
96             T c = maxc[sink];
97             sumc += c;
98             sumw += c * minw[sink];
99
100             int cur = sink;
101             while (cur != source) {
102                 Edge *e1 = prev[cur];
103                 e1->c -= c;
104                 Edge *e2 = &e[e1->v][e1->b];
105                 e2->c += c;
106                 cur = e2->v;
107             }
108         }
109     }
110     return make_pair(sumc, sumw);
111 }
112 };

```

7.10 AhoCorasick 自动机

Listing 104: `ac-automata.hpp`

```

1  #include <algorithm>
2  #include <queue>
3
4  using namespace std;
5
6  struct AhoCorasick {
7      static const int NONE = 0;
8      static const int MAXN = 1024;
9      static const int CHARSET = 26;
10
11      int end;
12      int tag[MAXN];
13      int fail[MAXN];

```

```

14     int trie[MAXN][CHARSET];
15
16     void init() {
17         tag[0] = NONE;
18         fill(trie[0], trie[0] + CHARSET, -1);
19         end = 1;
20     }
21
22     int add(int m, const int *s) {
23         int p = 0;
24         for (int i = 0; i < m; ++i) {
25             if (trie[p][*s] == -1) {
26                 tag[end] = NONE;
27                 fill(trie[end], trie[end] + CHARSET, -1);
28                 trie[p][*s] = end++;
29             }
30             p = trie[p][*s];
31             ++s;
32         }
33         return p;
34     }
35
36     void build(void) { // !!
37         queue<int> bfs;
38         fail[0] = 0;
39         for (int i = 0; i < CHARSET; ++i) {
40             if (trie[0][i] != -1) {
41                 fail[trie[0][i]] = 0;
42                 bfs.push(trie[0][i]);
43             } else {
44                 trie[0][i] = 0;
45             }
46         }
47         while (!bfs.empty()) {
48             int p = bfs.front();
49             tag[p] |= tag[fail[p]];
50             bfs.pop();
51             for (int i = 0; i < CHARSET; ++i) {
52                 if (trie[p][i] != -1) {
53                     fail[trie[p][i]] = trie[fail[p]][i];
54                     bfs.push(trie[p][i]);
55                 } else {
56                     trie[p][i] = trie[fail[p]][i];
57                 }
58             }
59         }
60     }
61 } ac;

```

7.11 后缀数组

Listing 105: **sa.hpp**

```

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         }
49         //
50         for (int i = 0; i < n; ++i) {
51             suffix[i] = make_pair(rank[i], i + m < n ? rank[i + m] : numeric_limits<int>::min());
52         }
53         for (int i = 0; i < n; ++i) {
54             if (i == 0 || suffix[sa[i]] != suffix[sa[i - 1]]) {
55                 rank[sa[i]] = i;
56             } else {
57                 rank[sa[i]] = rank[sa[i - 1]];
58             }
59         }
60     }
61
62     height.resize(n);
63     for (int i = 0, z = 0; i < n; ++i) {
64         if (rank[i] == 0) {
65             height[0] = z = 0;
66         } else {
67             int x = i, y = sa[rank[i] - 1];
68             z = max(0, z - 1);
69             while (x + z < n && y + z < n && a[x + z] == a[y + z]) {
70                 ++z;
71             }
72             height[rank[i]] = z;
73         }
74     }
75 }

```

```
76 };
```

7.12 LU 分解

Listing 106: **lu.hpp**

```
1  const int MAXN = 128;
2  const double EPS = 1e-10;
3
4  void LU(int n, double a[MAXN][MAXN], int r[MAXN], int c[MAXN]) {
5      for (int i = 0; i < n; ++i) {
6          r[i] = c[i] = i;
7      }
8      for (int k = 0; k < n; ++k) {
9          int ii = k, jj = k;
10         for (int i = k; i < n; ++i) {
11             for (int j = k; j < n; ++j) {
12                 if (fabs(a[i][j]) > fabs(a[ii][jj])) {
13                     ii = i;
14                     jj = j;
15                 }
16             }
17         }
18         swap(r[k], r[ii]);
19         swap(c[k], c[jj]);
20         for (int i = 0; i < n; ++i) {
21             swap(a[i][k], a[i][jj]);
22         }
23         for (int j = 0; j < n; ++j) {
24             swap(a[k][j], a[ii][j]);
25         }
26         if (fabs(a[k][k]) < EPS) {
27             continue;
28         }
29         for (int i = k + 1; i < n; ++i) {
30             a[i][k] = a[i][k] / a[k][k];
31             for (int j = k + 1; j < n; ++j) {
32                 a[i][j] -= a[i][k] * a[k][j];
33             }
34         }
35     }
36 }
37
38 void solve(int n, double a[MAXN][MAXN], int r[MAXN], int c[MAXN], double b[MAXN]) {
39     static double x[MAXN];
40     for (int i = 0; i < n; ++i) {
41         x[i] = b[r[i]];
42     }
43     for (int i = 0; i < n; ++i) {
44         for (int j = 0; j < i; ++j) {
45             x[i] -= a[i][j] * x[j];
46         }
47     }
48     for (int i = n - 1; i >= 0; --i) {
49         for (int j = n - 1; j > i; --j) {
50             x[i] -= a[i][j] * x[j];
51         }
52         if (fabs(a[i][i]) >= EPS) {
53             x[i] /= a[i][i];
54         } // else assert(fabs(x[i]) < EPS);
55     }
56     for (int i = 0; i < n; ++i) {
57         b[c[i]] = x[i];
58     }
```

```
58     }  
59 }  
60  
61 // LU(n - 1, a, r, c);  
62 // solve(n - 1, a, r, c, b);
```

8 对一类问题的处理方法