



Alfred
代码模版库

目录

1 比赛配置 and 奇技淫巧	3	5.5 二分图最大权匹配 (MaxAssignment, 基于 KM)	18
1.1 多组数据代码模板	3	5.6 一般图最大匹配 (Graph, 带花树算法)	20
1.2 快读快写	3	5.7 2-SAT	22
1.3 关闭流与 C 风格输入输出的同步	3	5.8 最大流	23
1.4 .clang-format	4	5.9 最小费用可行流 (或最大流)	24
1.5 debug.h	4	5.10 树链剖分	25
1.6 火车头	5	5.11 快速幂	27
1.7 c-cpp-properties.json	6	5.12 欧拉筛	28
1.8 launch.json	7	5.13 单点欧拉函数	28
1.9 settings.json	7	5.14 exgcd	28
1.10 tasks.json	7	5.15 组合数	29
2 数据结构	8	5.16 树状数组	29
2.1 珂朵莉树	8	5.17 Splay	30
2.2 树状数组	9	5.18 取模类, 按需写	33
2.3 静态可重区间信息 (支持 RMQ)	9	5.19 马拉车	36
2.4 PBDS 大常数平衡树	10	5.20 Z 函数	37
2.5 离散化容器	10	5.21 SA 后缀数组	37
2.6 并查集	11	6 Watashi 代码库 (备用)	38
2.7 出现次数统计	11	6.1 $O(n \log n) - O(1)$ RMQ	38
3 数学 (数论) 算法	12	6.2 $O(n \log n) - O(\log n)$ LCA	39
3.1 带模整数类	12	6.3 树状数组	40
3.2 计算几何	13	6.4 并查集	41
4 字符串算法	14	6.5 轻重权树剖分	41
4.1 字符串哈希	14	6.6 强连通分量	42
5 jiangly 代码库 (备用, 侵权请提出 issue)	15	6.7 双连通分量	43
5.1 int128 输出流自定义	15	6.8 二分图匹配	45
5.2 常用数学运算库函数及 gcd 重载	15	6.9 最小费用最大流	46
5.3 强连通分量缩点 (SCC)	15	6.10 AhoCorasick 自动机	48
5.4 割边与割边缩点 (EBCC)	16	6.11 后缀数组	49
		6.12 LU 分解	50
		7 对一类问题的处理方法	51

1 比赛配置 and 奇技淫巧

1.1 多组数据代码模板

Listing 1: template.cpp

```

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.cpp

```

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

```

1.3 关闭流与 C 风格输入输出的同步

Listing 3: io-sync-off.cpp

```

1  inline void optimizeIO(void) {
2      ios::sync_with_stdio(false);
3      cin.tie(NULL), cout.tie(NULL);
4  }

```

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

```

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

```

1.6 火车头

Listing 6: optimize-header.h

```

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
    shell再输入命令, 所以args还会经过shell再解析一遍
27 "group": {
28     "kind": "build",
29     "isDefault": true // 不为true时ctrl shift B就要手动选择了
30 },
31 "presentation": {
32     "echo": true,
33     "reveal": "always", // 执行任务时是否跳转到终端面板, 可以为always, silent, never。具体参见
        vsc的文档, 即使设为never, 手动点进去还是可以看到
34     "focus": false, // 设为true后可以使执行task时焦点聚集在终端, 但对编译C/C++来说, 设为true没
        有意义
35     "panel": "shared" // 不同的文件的编译信息共享一个终端面板
36 },
37 "problemMatcher": "$gcc" // 捕捉编译时终端里的报错信息到问题面板中, 修改代码后需要重新编译才会
    再次触发
38 // 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
39 }
40 ]
41 }

```

2 数据结构

2.1 珂朵莉树

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

Listing 11: chtholly.cpp

```

1  #include <set>
2
3  struct ChthollyTree {
4      typedef long long ll;
5      struct Node {
6          mutable ll l, r, v;
7          inline bool operator<(const Node &x) const { return l < x.l; }
8      };
9      std::set<Node> tr;
10     typedef std::set<Node>::iterator iterator;
11     ChthollyTree(void) = default;
12     ChthollyTree(int rng, int val) { init(rng, val); }
13     inline void init(ll rng, ll val) noexcept {
14         tr.insert({l, rng, val}), tr.insert({rng + 1, rng + 1, 0});
15     }
16     inline iterator begin(void) const noexcept { return tr.begin(); }
17     inline iterator end(void) const noexcept { return tr.end(); }
18     inline iterator split(ll pos) {
19         auto it = tr.lower_bound({pos, 0, 0});
20         if (it != tr.end() && it->l == pos) return it;
21         ll l = (---it)->l, r = it->r, v = it->v;
22         tr.erase(it), tr.insert({l, pos - 1, v});
23         return tr.insert({pos, r, v}).first;
24     }
25     inline void assign(ll l, ll r, ll v) {
26         auto R = split(r + 1), L = split(l);
27         tr.erase(L, R), tr.insert({l, r, v});

```



```

28     }
29     template <class _Functor> // func(iterator)
30     inline void modify(ll l, ll r, _Functor func) {
31         auto R = split(r + 1), L = split(l);
32         for (auto it = L; it != R; it++) func(it);
33     }
34     template <class _Functor> // func(ll &, iterator)
35     inline ll query(ll l, ll r, _Functor func) {
36         ll ans = 0;
37         auto R = split(r + 1);
38         for (auto it = split(l); it != R; it++) func(ans, it);
39         return ans;
40     }
41 };

```

2.2 树状数组

维护满足结合律且可差分信息的，常数较小的数据结构。

Listing 12: fenwick.cpp

```

1  template <class T>
2  struct Fenwick {
3      std::vector<T> c;
4      inline int lowbit(int x) { return x & -x; }
5      inline void merge(T &x, T &y) { x = x + y; }
6      inline T subtract(T x, T y) { return x - y; }
7      inline void update(size_t pos, T x) {
8          for (pos++; pos < c.size(); pos += lowbit(pos)) merge(c[pos], x);
9      }
10     inline void clear(void) {
11         for (auto &x : c) x = T();
12     }
13     inline T query(size_t pos) {
14         T ans = T();
15         for (pos++; pos; pos ^= lowbit(pos)) merge(ans, c[pos]);
16         return ans;
17     }
18     inline T query(size_t l, size_t r) {
19         return subtract(query(r), query(l - 1));
20     }
21     Fenwick(size_t len) : c(len + 2) {}
22 };

```

2.3 静态可重区间信息（支持 RMQ）

基于 ST 表，支持静态数组可重区间信息的数据结构。

Listing 13: sparse-table.cpp

```

1  template <class T>
2  struct MaxInfo {
3      T val;
4      MaxInfo() { val = T(); }
5      template <class InitT>
6      MaxInfo(InitT x) { val = x; }
7      MaxInfo operator+(MaxInfo &x) {
8          return {std::max(val, x.val)};
9      }
10 };
11 template <class T>
12 struct MinInfo {
13     T val;
14     MinInfo() { val = T(); }
15     template <class InitT>

```

```

16     MinInfo(InitT x) { val = x; }
17     MinInfo operator+(MinInfo &x) {
18         return {std::min(val, x.val)};
19     }
20 };
21 template <class T>
22 struct GcdInfo {
23     T val;
24     GcdInfo() { val = T(); }
25     template <class InitT>
26     GcdInfo(InitT x) { val = x; }
27     GcdInfo operator+(GcdInfo &x) {
28         return {std::gcd(val, x.val)};
29     }
30 };
31 template <class T>
32 struct SparseTable {
33 private:
34     int n;
35     std::vector<std::vector<T>> ST;
36
37 public:
38     SparseTable() {}
39     SparseTable(int N) : n(N), ST(N, std::vector<T>(__lg(N) + 1)) {}
40     template <class InitT>
41     SparseTable(std::vector<InitT> init) : SparseTable(init.size()) {
42         for (int i = 0; i < n; i++) ST[i][0] = T(init[i]);
43         for (int i = 1; (1 << i) <= n; i++) {
44             for (int j = 0; j + (1 << i) - 1 < n; j++) {
45                 ST[j][i] = ST[j][i - 1] + ST[j + (1 << i) - 1][i - 1];
46             }
47         }
48     }
49     inline T query(int l, int r) { // 0 based
50         int w = std::__lg(r - l + 1);
51         return ST[l][w] + ST[r - (1 << w) + 1][w];
52     }
53 };

```

2.4 PBDS 大常数平衡树

GNU PBDS 提供的大常数基于 rb-tree 的平衡树。

Listing 14: pbds-balance-tree.cpp

```

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

```

2.5 离散化容器

Listing 15: discretization.cpp

```

1 #include <vector>
2
3 template <class _Tp>
4 struct Mess {
5     std::vector<_Tp> v;
6     bool initialized = false;

```

```

7     inline _Tp origin(int idx) { return v[idx - 1]; }
8     inline void insert(_Tp x) { v.push_back(x); }
9     template <typename T, typename... V>
10    inline void insert(T x, V... v) { insert(x), insert(v...); }
11    inline void init(void) {
12        sort(v.begin(), v.end()), initialized = true;
13        v.erase(unique(v.begin(), v.end()), v.end());
14    }
15    inline int query(_Tp x) {
16        if (!initialized) init();
17        return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
18    }
19    inline bool exist(_Tp x) { return origin(query(x)) == x; }
20 };

```

2.6 并查集

Listing 16: dsu.cpp

```

1 struct DSU {
2     std::vector<int> fa;
3     DSU(int n) : fa(n + 1) {
4         iota(fa.begin(), fa.end(), 0);
5     }
6     inline int find(int x) {
7         return fa[x] == x ? x : fa[x] = find(fa[x]);
8     }
9     inline void merge(int x, int y) {
10        int fx = find(x), fy = find(y);
11        if (fx != fy) fa[fx] = fy;
12    }
13 };

```

2.7 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 17: appear-statistics.cpp

```

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

```

```

27     std::vector<std::vector<int>> pos;
28
29 public:
30     AppearStats(void) : n(0) {}
31     AppearStats(std::vector<T> &init) : n(init.size()) { _init(init); }
32     inline void _init(std::vector<T> &init) {
33         for (auto item : init) M.insert(item);
34         n = init.size(), M.init(), pos.resize(M.v.size());
35         for (size_t i = 0; i < n; i++) {
36             pos[M.query(init[i]) - 1].push_back(i);
37         }
38     }
39     // Use [base] as the beginning of index, return -1 if x doesn't exist.
40     inline int first(int l, int r, T x, int base = 0) {
41         l -= base, r -= base;
42         if (!M.exist(x)) return -1;
43         std::vector<int> &P = pos[M.query(x) - 1];
44         auto it = std::lower_bound(P.begin(), P.end(), l);
45         return it == P.end() || *it > r ? -1 : *it + base;
46     }
47     // Use [base] as the beginning of index, return -1 if x doesn't exist.
48     inline int last(int l, int r, T x, int base = 0) {
49         l -= base, r -= base;
50         if (!M.exist(x)) return -1;
51         std::vector<int> &P = pos[M.query(x) - 1];
52         auto it = std::upper_bound(P.begin(), P.end(), r);
53         return it == P.begin() || *std::prev(it) < l ? -1 : *std::prev(it) + base;
54     }
55     inline int count(int l, int r, T x, int base = 0) {
56         l -= base, r -= base;
57         if (!M.exist(x)) return 0;
58         std::vector<int> &P = pos[M.query(x) - 1];
59         auto L = std::lower_bound(P.begin(), P.end(), l);
60         auto R = std::upper_bound(P.begin(), P.end(), r);
61         if (L == P.end() || R == P.begin()) return 0;
62         if (*L > r || *std::prev(R) < l) return 0;
63         return R - L;
64     }
65 };

```

3 数学（数论）算法

3.1 带模整数类

Listing 18: mod-int.cpp

```

1  template <int mod>
2  inline int64_t down(int64_t x) { return x >= mod ? x - mod : x; }
3  template <int mod>
4  struct ModInt {
5      int64_t x;
6      ModInt() = default;
7      ModInt(int64_t x) : x((x % mod + mod) % mod) {}
8      friend istream &operator>>(istream &in, ModInt &a) { return in >> a.x; }
9      friend ostream &operator<<(ostream &out, ModInt a) { return out << a.x; }
10     friend ModInt operator+(ModInt a, ModInt b) { return down<mod>(a.x + b.x); }
11     friend ModInt operator-(ModInt a, ModInt b) { return down<mod>(a.x - b.x + mod); }
12     friend ModInt operator*(ModInt a, ModInt b) { return (_int128)a.x * b.x % mod; }
13     friend ModInt operator/(ModInt a, ModInt b) { return a * ~b; }
14     friend ModInt operator^(ModInt a, long long b) {
15         ModInt ans = 1;
16         for (; b >= 1, a *= a)
17             if (b & 1) ans *= a;
18         return ans;
19     }

```

```

20 friend ModInt operator~(ModInt a) { return a ^ (mod - 2); }
21 friend ModInt operator-(ModInt a) { return down<mod>(mod - a.x); }
22 friend ModInt &operator+=(ModInt &a, ModInt b) { return a = a + b; }
23 friend ModInt &operator-=(ModInt &a, ModInt b) { return a = a - b; }
24 friend ModInt &operator*=(ModInt &a, ModInt b) { return a = a * b; }
25 friend ModInt &operator/=(ModInt &a, ModInt b) { return a = a / b; }
26 friend ModInt &operator^=(ModInt &a, long long b) { return a = a ^ b; }
27 friend ModInt &operator++(ModInt &a) { return a += 1; }
28 friend ModInt operator++(ModInt &a, int) {
29     ModInt x = a;
30     a += 1;
31     return x;
32 }
33 friend ModInt &operator--(ModInt &a) { return a -= 1; }
34 friend ModInt operator--(ModInt &a, int) {
35     ModInt x = a;
36     a -= 1;
37     return x;
38 }
39 friend bool operator==(ModInt a, ModInt b) { return a.x == b.x; }
40 friend bool operator!=(ModInt a, ModInt b) { return !(a == b); }
41 };
42 using mint = ModInt<>;

```

3.2 计算几何

Listing 19: computation-geometry.cpp

```

1 template <class T>
2 struct Point {
3     T x, y;
4     Point(void) = default;
5     Point(T X, T Y) : x(X), y(Y) {}
6     inline bool operator==(const Point B) {
7         return x == B.x && y == B.y;
8     }
9     friend std::ostream &operator<<(std::ostream &out, Point P) {
10         return out << "(" << P.x << ", " << P.y << ")";
11     }
12     friend std::istream &operator>>(std::istream &in, Point &P) {
13         return in >> P.x >> P.y;
14     }
15 };
16 template <class T>
17 struct Line {
18     T A, B, C; // Ax + By + C = 0
19     Line(void) = default;
20     Line(T a, T b, T c) : A(a), B(b), C(c) {} // Ax + By + C = 0
21     Line(T k, T b) : A(k), B(-1), C(b) {} // y = kx + b
22 };
23 template <class T>
24 inline int sign(T x) {
25     return x == 0 ? 0 : (x < 0 ? -1 : 1);
26 }
27 template <class T>
28 inline bool parallel(Line<T> P, Line<T> Q) {
29     return P.A * Q.B == P.B * Q.A;
30 }
31 template <class T>
32 inline Point<T> intersect(Line<T> P, Line<T> Q) {
33     assert(!parallel(P, Q));
34     return Point<T>{
35         (P.C * Q.B - Q.C * P.B) / (Q.A * P.B - P.A * Q.B),
36         (P.C * Q.A - Q.C * P.A) / (P.A * Q.B - Q.A * P.B)
37     };
38 }

```

```

39 template <class T>
40 inline Line<T> getLine(Point<T> P, Point<T> Q) {
41     assert(!(P == Q));
42     if (P.x == Q.x) {
43         return Line<T>(-1, 0, P.x);
44     } else if (P.y == Q.y) {
45         return Line<T>(0, -1, P.y);
46     } else {
47         const T k = (P.x - Q.x) / (P.y - Q.y);
48         return Line<T>(k, P.y - k * P.x);
49     }
50 }
51 template <class T>
52 inline bool pointOnLine(Point<T> P, Line<T> L) {
53     return L.A * P.x + L.B * P.y + L.C == 0;
54 }

```

4 字符串算法

4.1 字符串哈希

Listing 20: hashed-string.cpp

```

1  using i64 = long long;
2  using i128 = __int128;
3  class HashedString {
4  private:
5      // change M and B if you want
6      static const i64 M = (1LL << 61) - 1;
7      static const i64 B;
8      // pow[i] contains B^i % M
9      static std::vector<i64> pow;
10     // p_hash[i] is the hash of the first i characters of the given string
11     std::vector<i64> r_hash, p_hash;
12     i128 mul(i64 a, i64 b) { return (i128)a * b; }
13     i64 mod_mul(i64 a, i64 b) { return mul(a, b) % M; }
14
15 public:
16     HashedString(const string &s) : r_hash(s.size() + 1), p_hash(s.size() + 1) {
17         while (pow.size() < s.size()) { pow.push_back(mod_mul(pow.back(), B)); }
18         p_hash[0] = 0;
19         r_hash[0] = 0;
20         for (size_t i = 0; i < s.size(); i++) {
21             p_hash[i + 1] = (mul(p_hash[i], B) + s[i]) % M; // 1-based
22         }
23         i64 sz = s.size();
24         for (int i = sz - 1, j = 0; i >= 0; i--, j++) {
25             r_hash[j + 1] = (mul(r_hash[j], B) + s[i]) % M;
26         }
27     }
28     i64 getHash(int start, int end) { // 0 based
29         i64 raw_val = p_hash[end + 1] - mod_mul(p_hash[start], pow[end - start + 1]);
30         return (raw_val + M) % M;
31     }
32     i64 getRHash(int start, int end) { // 0 based
33         i64 raw_val = r_hash[end + 1] - mod_mul(r_hash[start], pow[end - start + 1]);
34         return (raw_val + M) % M;
35     }
36 };
37 std::vector<i64> HashedString::pow = {1};
38 mt19937 rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
39 const i64 HashedString::B = uniform_int_distribution<i64>(0, M - 1)(rng);

```

5 jiangly 代码库 (备用, 侵权请提出 issue)

5.1 int128 输出流自定义

Listing 21: others/i128-stream.cpp

```

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

```

5.2 常用数学运算库函数及 gcd 重载

Listing 22: others/clf.cpp

```

1  using i64 = long long;
2  using i128 = __int128;
3  inline i64 ceilDiv(i64 n, i64 m) {
4      if (n >= 0) {
5          return (n + m - 1) / m;
6      } else {
7          return n / m;
8      }
9  }
10 inline i64 floorDiv(i64 n, i64 m) {
11     if (n >= 0) {
12         return n / m;
13     } else {
14         return (n - m + 1) / m;
15     }
16 }
17 template <class T>
18 inline void chmax(T &a, T b) {
19     if (a < b) a = b;
20 }
21 template <class T>
22 inline void chmin(T &a, T b) {
23     if (!(a < b)) a = b;
24 }
25 inline i128 gcd(i128 a, i128 b) {
26     return b ? gcd(b, a % b) : a;
27 }

```

5.3 强连通分量缩点 (SCC)

Listing 23: graph/scc.cpp

```

1  #include <vector>
2
3  struct SCC {
4      int n;
5      std::vector<std::vector<int>>> adj;
6      std::vector<int> stk;
7      std::vector<int> dfn, low, bel;
8      int cur, cnt;
9
10     SCC() {}
11     SCC(int n) {
12         init(n);
13     }
14
15     void init(int n) {
16         this->n = n;
17         adj.assign(n, {});
18         dfn.assign(n, -1);
19         low.resize(n);
20         bel.assign(n, -1);
21         stk.clear();
22         cur = cnt = 0;
23     }
24
25     void addEdge(int u, int v) {
26         adj[u].push_back(v);
27     }
28
29     void dfs(int x) {
30         dfn[x] = low[x] = cur++;
31         stk.push_back(x);
32
33         for (auto y : adj[x]) {
34             if (dfn[y] == -1) {
35                 dfs(y);
36                 low[x] = std::min(low[x], low[y]);
37             } else if (bel[y] == -1) {
38                 low[x] = std::min(low[x], dfn[y]);
39             }
40         }
41
42         if (dfn[x] == low[x]) {
43             int y;
44             do {
45                 y = stk.back();
46                 bel[y] = cnt;
47                 stk.pop_back();
48             } while (y != x);
49             cnt++;
50         }
51     }
52
53     std::vector<int> work() {
54         for (int i = 0; i < n; i++) {
55             if (dfn[i] == -1) {
56                 dfs(i);
57             }
58         }
59         return bel;
60     }
61 };

```

5.4 割边与割边缩点 (EBCC)

Listing 24: graph/ebcc.cpp

```

1  #include <set>
2  #include <vector>
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 };

```

5.5 二分图最大权匹配 (MaxAssignment, 基于 KM)

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

```

1  #include <queue>
2  #include <vector>
3
4  template <class T>
5  struct MaxAssignment {
6  public:
7      T solve(int nx, int ny, std::vector<std::vector<T>> a) {
8          assert(0 <= nx && nx <= ny);
9          assert(int(a.size()) == nx);
10         for (int i = 0; i < nx; ++i) {
11             assert(int(a[i].size()) == ny);
12             for (auto x : a[i])
13                 assert(x >= 0);
14         }
15
16         auto update = [&](int x) {
17             for (int y = 0; y < ny; ++y) {
18                 if (lx[x] + ly[y] - a[x][y] < slack[y]) {
19                     slack[y] = lx[x] + ly[y] - a[x][y];
20                     slackx[y] = x;
21                 }
22             }
23         };
24
25         costs.resize(nx + 1);
26         costs[0] = 0;
27         lx.assign(nx, std::numeric_limits<T>::max());
28         ly.assign(ny, 0);
29         xy.assign(nx, -1);
30         yx.assign(ny, -1);
31         slackx.resize(ny);
32         for (int cur = 0; cur < nx; ++cur) {
33             std::queue<int> que;
34             visx.assign(nx, false);
35             visy.assign(ny, false);
36             slack.assign(ny, std::numeric_limits<T>::max());

```

```

37         p.assign(nx, -1);
38
39         for (int x = 0; x < nx; ++x) {
40             if (xy[x] == -1) {
41                 que.push(x);
42                 visx[x] = true;
43                 update(x);
44             }
45         }
46
47         int ex, ey;
48         bool found = false;
49         while (!found) {
50             while (!que.empty() && !found) {
51                 auto x = que.front();
52                 que.pop();
53                 for (int y = 0; y < ny; ++y) {
54                     if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
55                         if (yx[y] == -1) {
56                             ex = x;
57                             ey = y;
58                             found = true;
59                             break;
60                         }
61                         que.push(yx[y]);
62                         p[yx[y]] = x;
63                         visy[y] = visx[yx[y]] = true;
64                         update(yx[y]);
65                     }
66                 }
67             }
68             if (found)
69                 break;
70
71             T delta = std::numeric_limits<T>::max();
72             for (int y = 0; y < ny; ++y)
73                 if (!visy[y])
74                     delta = std::min(delta, slack[y]);
75             for (int x = 0; x < nx; ++x)
76                 if (visx[x])
77                     lx[x] -= delta;
78             for (int y = 0; y < ny; ++y) {
79                 if (visy[y]) {
80                     ly[y] += delta;
81                 } else {
82                     slack[y] -= delta;
83                 }
84             }
85             for (int y = 0; y < ny; ++y) {
86                 if (!visy[y] && slack[y] == 0) {
87                     if (yx[y] == -1) {
88                         ex = slackx[y];
89                         ey = y;
90                         found = true;
91                         break;
92                     }
93                     que.push(yx[y]);
94                     p[yx[y]] = slackx[y];
95                     visy[y] = visx[yx[y]] = true;
96                     update(yx[y]);
97                 }
98             }
99         }
100
101         costs[cur + 1] = costs[cur];
102         for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {

```

```

103         costs[cur + 1] += a[x][y];
104         if (xy[x] != -1)
105             costs[cur + 1] -= a[x][xy[x]];
106         ty = xy[x];
107         xy[x] = y;
108         yx[y] = x;
109     }
110 }
111 return costs[nx];
112 }
113 std::vector<int> assignment() {
114     return xy;
115 }
116 std::pair<std::vector<T>, std::vector<T>> labels() {
117     return std::make_pair(lx, ly);
118 }
119 std::vector<T> weights() {
120     return costs;
121 }
122
123 private:
124     std::vector<T> lx, ly, slack, costs;
125     std::vector<int> xy, yx, p, slackx;
126     std::vector<bool> visx, visy;
127 };

```

5.6 一般图最大匹配 (Graph, 带花树算法)

Listing 26: graph/general-match.cpp

```

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

```

```

103         match[v] = u;
104         break;
105     }
106 }
107 }
108 };
109
110 greedy();
111
112 for (int u = 0; u < n; ++u)
113     if (match[u] == -1)
114         augment(u);
115
116 return match;
117 }
118 };

```

5.7 2-SAT

Listing 27: graph/2-sat.cpp

```

1  #include <vector>
2
3  struct TwoSat {
4      int n;
5      std::vector<std::vector<int>>> e;
6      std::vector<bool> ans;
7      TwoSat(int n) : n(n), e(2 * n), ans(n) {}
8      void addClause(int u, bool f, int v, bool g) {
9          e[2 * u + !f].push_back(2 * v + g);
10         e[2 * v + !g].push_back(2 * u + f);
11     }
12     bool satisfiable() {
13         std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
14         std::vector<int> stk;
15         int now = 0, cnt = 0;
16         std::function<void(int)> tarjan = [&](int u) {
17             stk.push_back(u);
18             dfn[u] = low[u] = now++;
19             for (auto v : e[u]) {
20                 if (dfn[v] == -1) {
21                     tarjan(v);
22                     low[u] = std::min(low[u], low[v]);
23                 } else if (id[v] == -1) {
24                     low[u] = std::min(low[u], dfn[v]);
25                 }
26             }
27             if (dfn[u] == low[u]) {
28                 int v;
29                 do {
30                     v = stk.back();
31                     stk.pop_back();
32                     id[v] = cnt;
33                 } while (v != u);
34                 ++cnt;
35             }
36         };
37         for (int i = 0; i < 2 * n; ++i)
38             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 };

5.8 最大流

Listing 28: graph/max-flow.cpp

```

1  constexpr int inf = 1E9;
2  template <class T>
3  struct MaxFlow {
4      struct _Edge {
5          int to;
6          T cap;
7          _Edge(int to, T cap) : to(to), cap(cap) {}
8      };
9
10     int n;
11     std::vector<_Edge> e;
12     std::vector<std::vector<int>> g;
13     std::vector<int> cur, h;
14
15     MaxFlow() {}
16     MaxFlow(int n) {
17         init(n);
18     }
19
20     void init(int n) {
21         this->n = n;
22         e.clear();
23         g.assign(n, {});
24         cur.resize(n);
25         h.resize(n);
26     }
27
28     bool bfs(int s, int t) {
29         h.assign(n, -1);
30         std::queue<int> que;
31         h[s] = 0;
32         que.push(s);
33         while (!que.empty()) {
34             const int u = que.front();
35             que.pop();
36             for (int i : g[u]) {
37                 auto [v, c] = e[i];
38                 if (c > 0 && h[v] == -1) {
39                     h[v] = h[u] + 1;
40                     if (v == t) {
41                         return true;
42                     }
43                     que.push(v);
44                 }
45             }
46         }
47         return false;
48     }
49
50     T dfs(int u, int t, T f) {
51         if (u == t) {
52             return f;
53         }
54         auto r = f;
55         for (int &i = cur[u]; i < int(g[u].size()); ++i) {
56             const int j = g[u][i];
57             auto [v, c] = e[j];
58             if (c > 0 && h[v] == h[u] + 1) {
59                 auto a = dfs(v, t, std::min(r, c));
60                 e[j].cap -= a;

```

```

61         e[j ^ 1].cap += a;
62         r -= a;
63         if (r == 0) {
64             return f;
65         }
66     }
67 }
68 return f - r;
69 }
70 void addEdge(int u, int v, T c) {
71     g[u].push_back(e.size());
72     e.emplace_back(v, c);
73     g[v].push_back(e.size());
74     e.emplace_back(u, 0);
75 }
76 T flow(int s, int t) {
77     T ans = 0;
78     while (bfs(s, t)) {
79         cur.assign(n, 0);
80         ans += dfs(s, t, std::numeric_limits<T>::max());
81     }
82     return ans;
83 }
84
85 std::vector<bool> minCut() {
86     std::vector<bool> c(n);
87     for (int i = 0; i < n; i++) {
88         c[i] = (h[i] != -1);
89     }
90     return c;
91 }
92
93 struct Edge {
94     int from;
95     int to;
96     T cap;
97     T flow;
98 };
99 std::vector<Edge> edges() {
100     std::vector<Edge> a;
101     for (int i = 0; i < e.size(); i += 2) {
102         Edge x;
103         x.from = e[i + 1].to;
104         x.to = e[i].to;
105         x.cap = e[i].cap + e[i + 1].cap;
106         x.flow = e[i + 1].cap;
107         a.push_back(x);
108     }
109     return a;
110 }
111 };

```

5.9 最小费用可行流 (或最大流)

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

```

1 struct MCFGGraph {
2     struct Edge {
3         int v, c, f;
4         Edge(int v, int c, int f) : v(v), c(c), f(f) {}
5     };
6     const int n;
7     std::vector<Edge> e;
8     std::vector<std::vector<int>> g;
9     std::vector<i64> h, dis;
10    std::vector<int> pre;

```



```

11  bool dijkstra(int s, int t) {
12      dis.assign(n, std::numeric_limits<i64>::max());
13      pre.assign(n, -1);
14      std::priority_queue<std::pair<i64, int>, std::vector<std::pair<i64, int>>, std::greater<std::pair<
        i64, int>>> que;
15      dis[s] = 0;
16      que.emplace(0, s);
17      while (!que.empty()) {
18          i64 d = que.top().first;
19          int u = que.top().second;
20          que.pop();
21          if (dis[u] < d) continue;
22          for (int i : g[u]) {
23              int v = e[i].v;
24              int c = e[i].c;
25              int f = e[i].f;
26              if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
27                  dis[v] = d + h[u] - h[v] + f;
28                  pre[v] = i;
29                  que.emplace(dis[v], v);
30              }
31          }
32      }
33      return dis[t] != std::numeric_limits<i64>::max();
34  }
35  MCFGraph(int n) : n(n), g(n) {}
36  void addEdge(int u, int v, int c, int f) { // 可行流
37      if (f < 0) {
38          g[u].push_back(e.size());
39          e.emplace_back(v, 0, f);
40          g[v].push_back(e.size());
41          e.emplace_back(u, c, -f);
42      } else {
43          g[u].push_back(e.size());
44          e.emplace_back(v, c, f);
45          g[v].push_back(e.size());
46          e.emplace_back(u, 0, -f);
47      }
48  }
49  // void addEdge(int u, int v, int c, int f) { // 最大流
50  //     g[u].push_back(e.size());
51  //     e.emplace_back(v, c, f);
52  //     g[v].push_back(e.size());
53  //     e.emplace_back(u, 0, -f);
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;
66          }
67          flow += aug;
68          cost += i64(aug) * h[t];
69      }
70      return std::make_pair(flow, cost);
71  }
72 };

```

5.10 树链剖分

Listing 30: graph/hld.cpp

```

1  struct HLD {
2      int n;
3      std::vector<int> siz, top, dep, parent, in, out, seq;
4      std::vector<std::vector<int>>> adj;
5      int cur;
6
7      HLD() {}
8      HLD(int n) {
9          init(n);
10     }
11     void init(int n) {
12         this->n = n;
13         siz.resize(n);
14         top.resize(n);
15         dep.resize(n);
16         parent.resize(n);
17         in.resize(n);
18         out.resize(n);
19         seq.resize(n);
20         cur = 0;
21         adj.assign(n, {});
22     }
23     void addEdge(int u, int v) {
24         adj[u].push_back(v);
25         adj[v].push_back(u);
26     }
27     void work(int root = 0) {
28         top[root] = root;
29         dep[root] = 0;
30         parent[root] = -1;
31         dfs1(root);
32         dfs2(root);
33     }
34     void dfs1(int u) {
35         if (parent[u] != -1) {
36             adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]));
37         }
38
39         siz[u] = 1;
40         for (auto &v : adj[u]) {
41             parent[v] = u;
42             dep[v] = dep[u] + 1;
43             dfs1(v);
44             siz[u] += siz[v];
45             if (siz[v] > siz[adj[u][0]]) {
46                 std::swap(v, adj[u][0]);
47             }
48         }
49     }
50     void dfs2(int u) {
51         in[u] = cur++;
52         seq[in[u]] = u;
53         for (auto v : adj[u]) {
54             top[v] = v == adj[u][0] ? top[u] : v;
55             dfs2(v);
56         }
57         out[u] = cur;
58     }
59     int lca(int u, int v) {
60         while (top[u] != top[v]) {
61             if (dep[top[u]] > dep[top[v]]) {
62                 u = parent[top[u]];
63             } else {
64                 v = parent[top[v]];
65             }
66         }
67     }

```

```

66     }
67     return dep[u] < dep[v] ? u : v;
68 }
69
70 int dist(int u, int v) {
71     return dep[u] + dep[v] - 2 * dep[lca(u, v)];
72 }
73
74 int jump(int u, int k) {
75     if (dep[u] < k) {
76         return -1;
77     }
78
79     int d = dep[u] - k;
80
81     while (dep[top[u]] > d) {
82         u = parent[top[u]];
83     }
84
85     return seq[in[u] - dep[u] + d];
86 }
87
88 bool isAncestor(int u, int v) {
89     return in[u] <= in[v] && in[v] < out[u];
90 }
91
92 int rootedParent(int u, int v) {
93     std::swap(u, v);
94     if (u == v) {
95         return u;
96     }
97     if (!isAncestor(u, v)) {
98         return parent[u];
99     }
100     auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
101         return in[x] < in[y];
102     }) - 1;
103     return *it;
104 }
105
106 int rootedSize(int u, int v) {
107     if (u == v) {
108         return n;
109     }
110     if (!isAncestor(v, u)) {
111         return siz[v];
112     }
113     return n - siz[rootedParent(u, v)];
114 }
115
116 int rootedLca(int a, int b, int c) {
117     return lca(a, b) ^ lca(b, c) ^ lca(c, a);
118 }
119 };

```

5.11 快速幂

Listing 31: math/fast-pow.cpp

```

1 int power(int a, i64 b, int p) {
2     int res = 1;
3     for (; b; b /= 2, a = 1LL * a * a % p) {
4         if (b % 2) {
5             res = 1LL * res * a % p;
6         }
7     }

```

```

8     return res;
9 }

```

5.12 欧拉筛

Listing 32: math/euler-sieve.cpp

```

1  std::vector<int> minp, primes;
2
3  void sieve(int n) {
4      minp.assign(n + 1, 0);
5      primes.clear();
6
7      for (int i = 2; i <= n; i++) {
8          if (minp[i] == 0) {
9              minp[i] = i;
10             primes.push_back(i);
11         }
12
13         for (auto p : primes) {
14             if (i * p > n) {
15                 break;
16             }
17             minp[i * p] = p;
18             if (p == minp[i]) {
19                 break;
20             }
21         }
22     }
23 }

```

5.13 单点欧拉函数

Listing 33: math/phi.cpp

```

1  int phi(int n) {
2      int res = n;
3      for (int i = 2; i * i <= n; i++) {
4          if (n % i == 0) {
5              while (n % i == 0) {
6                  n /= i;
7              }
8              res = res / i * (i - 1);
9          }
10     }
11     if (n > 1) {
12         res = res / n * (n - 1);
13     }
14     return res;
15 }

```

5.14 exgcd

Listing 34: math/exgcd.cpp

```

1  int exgcd(int a, int b, int &x, int &y) {
2      if (!b) {
3          x = 1, y = 0;
4          return a;
5      }
6      int g = exgcd(b, a % b, y, x);
7      y -= a / b * x;
8      return g;
9 }

```

5.15 组合数

Listing 35: math/comb.cpp

```

1  struct Comb {
2      int n;
3      std::vector<Z> _fac;
4      std::vector<Z> _invfac;
5      std::vector<Z> _inv;
6
7      Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
8      Comb(int n) : Comb() {
9          init(n);
10     }
11
12     void init(int m) {
13         m = std::min(m, Z::getMod() - 1);
14         if (m <= n) return;
15         _fac.resize(m + 1);
16         _invfac.resize(m + 1);
17         _inv.resize(m + 1);
18
19         for (int i = n + 1; i <= m; i++) {
20             _fac[i] = _fac[i - 1] * i;
21         }
22         _invfac[m] = _fac[m].inv();
23         for (int i = m; i > n; i--) {
24             _invfac[i - 1] = _invfac[i] * i;
25             _inv[i] = _invfac[i] * _fac[i - 1];
26         }
27         n = m;
28     }
29
30     Z fac(int m) {
31         if (m > n) init(2 * m);
32         return _fac[m];
33     }
34     Z invfac(int m) {
35         if (m > n) init(2 * m);
36         return _invfac[m];
37     }
38     Z inv(int m) {
39         if (m > n) init(2 * m);
40         return _inv[m];
41     }
42     Z binom(int n, int m) {
43         if (n < m || m < 0) return 0;
44         return fac(n) * invfac(m) * invfac(n - m);
45     }
46 } comb;

```

5.16 树状数组

Listing 36: ds/fenwick.cpp

```

1  template <typename T>
2  struct Fenwick {
3      int n;
4      std::vector<T> a;
5
6      Fenwick(int n_ = 0) {
7          init(n_);
8      }
9
10     void init(int n_) {
11         n = n_;

```

```

12     a.assign(n, T{});
13 }
14
15 void add(int x, const T &v) {
16     for (int i = x + 1; i <= n; i += i & -i) {
17         a[i - 1] = a[i - 1] + v;
18     }
19 }
20
21 T sum(int x) {
22     T ans{};
23     for (int i = x; i > 0; i -= i & -i) {
24         ans = ans + a[i - 1];
25     }
26     return ans;
27 }
28
29 T rangeSum(int l, int r) {
30     return sum(r) - sum(l);
31 }
32
33 int select(const T &k) {
34     int x = 0;
35     T cur{};
36     for (int i = 1 << std::__lg(n); i; i /= 2) {
37         if (x + i <= n && cur + a[x + i - 1] <= k) {
38             x += i;
39             cur = cur + a[x - 1];
40         }
41     }
42     return x;
43 }
44 };

```

5.17 Splay

Listing 37: ds/splay.cpp

```

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 ? 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 }

```

5.18 取模类, 按需写

Listing 38: ds/mod.cpp

```

1  template <class T>
2  constexpr T power(T a, i64 b) {
3      T res = 1;
4      for (; b; b /= 2, a *= a) {
5          if (b % 2) {
6              res *= a;
7          }
8      }
9      return res;
10 }
11
12 constexpr i64 mul(i64 a, i64 b, i64 p) {
13     i64 res = a * b - i64(1.L * a * b / p) * p;
14     res %= p;
15     if (res < 0) {
16         res += p;
17     }
18     return res;
19 }
20 template <i64 P>
21 struct MLong {
22     i64 x;
23     constexpr MLong() : x{} {}
24     constexpr MLong(i64 x) : x{norm(x % getMod())} {}
25
26     static i64 Mod;
27     constexpr static i64 getMod() {
28         if (P > 0) {
29             return P;

```

```

30     } else {
31         return Mod;
32     }
33 }
34 constexpr static void setMod(i64 Mod_) {
35     Mod = Mod_;
36 }
37 constexpr i64 norm(i64 x) const {
38     if (x < 0) {
39         x += getMod();
40     }
41     if (x >= getMod()) {
42         x -= getMod();
43     }
44     return x;
45 }
46 constexpr i64 val() const {
47     return x;
48 }
49 explicit constexpr operator i64() const {
50     return x;
51 }
52 constexpr MLong operator-() const {
53     MLong res;
54     res.x = norm(getMod() - x);
55     return res;
56 }
57 constexpr MLong inv() const {
58     assert(x != 0);
59     return power(*this, getMod() - 2);
60 }
61 constexpr MLong &operator*=(MLong rhs) & {
62     x = mul(x, rhs.x, getMod());
63     return *this;
64 }
65 constexpr MLong &operator+=(MLong rhs) & {
66     x = norm(x + rhs.x);
67     return *this;
68 }
69 constexpr MLong &operator-=(MLong rhs) & {
70     x = norm(x - rhs.x);
71     return *this;
72 }
73 constexpr MLong &operator/=(MLong rhs) & {
74     return *this *= rhs.inv();
75 }
76 friend constexpr MLong operator*(MLong lhs, MLong rhs) {
77     MLong res = lhs;
78     res *= rhs;
79     return res;
80 }
81 friend constexpr MLong operator+(MLong lhs, MLong rhs) {
82     MLong res = lhs;
83     res += rhs;
84     return res;
85 }
86 friend constexpr MLong operator-(MLong lhs, MLong rhs) {
87     MLong res = lhs;
88     res -= rhs;
89     return res;
90 }
91 friend constexpr MLong operator/(MLong lhs, MLong rhs) {
92     MLong res = lhs;
93     res /= rhs;
94     return res;
95 }

```

```

96     friend constexpr std::istream &operator>>(std::istream &is, MLong &a) {
97         i64 v;
98         is >> v;
99         a = MLong(v);
100        return is;
101    }
102    friend constexpr std::ostream &operator<<(std::ostream &os, const MLong &a) {
103        return os << a.val();
104    }
105    friend constexpr bool operator==(MLong lhs, MLong rhs) {
106        return lhs.val() == rhs.val();
107    }
108    friend constexpr bool operator!=(MLong lhs, MLong rhs) {
109        return lhs.val() != rhs.val();
110    }
111 };
112
113 template <>
114 i64 MLong<OLL>::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>;

```

5.19 马拉车

Listing 39: string/manacher.cpp

```

1  std::vector<int> manacher(std::vector<int> s) {
2      std::vector<int> t{0};
3      for (auto c : s) {
4          t.push_back(c);
5          t.push_back(0);
6      }

```

```

7     int n = t.size();
8     std::vector<int> r(n);
9     for (int i = 0, j = 0; i < n; i++) {
10         if (2 * j - i >= 0 && j + r[j] > i) {
11             r[i] = std::min(r[2 * j - i], j + r[j] - i);
12         }
13         while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
14             r[i] += 1;
15         }
16         if (i + r[i] > j + r[j]) {
17             j = i;
18         }
19     }
20     return r;
21 }

```

5.20 Z 函数

Listing 40: string/z-func.cpp

```

1  std::vector<int> zFunction(std::string s) {
2      int n = s.size();
3      std::vector<int> z(n + 1);
4      z[0] = n;
5      for (int i = 1, j = 1; i < n; i++) {
6          z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
7          while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
8              z[i]++;
9          }
10         if (i + z[i] > j + z[j]) {
11             j = i;
12         }
13     }
14     return z;
15 }

```

5.21 SA 后缀数组

Listing 41: string/suffix-array.cpp

```

1  struct SuffixArray {
2      int n;
3      std::vector<int> sa, rk, lc;
4      SuffixArray(const std::string &s) {
5          n = s.length();
6          sa.resize(n);
7          lc.resize(n - 1);
8          rk.resize(n);
9          std::iota(sa.begin(), sa.end(), 0);
10         std::sort(sa.begin(), sa.end(), [&](int a, int b) { return s[a] < s[b]; });
11         rk[sa[0]] = 0;
12         for (int i = 1; i < n; ++i)
13             rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
14         int k = 1;
15         std::vector<int> tmp, cnt(n);
16         tmp.reserve(n);
17         while (rk[sa[n - 1]] < n - 1) {
18             tmp.clear();
19             for (int i = 0; i < k; ++i)
20                 tmp.push_back(n - k + i);
21             for (auto i : sa)
22                 if (i >= k)
23                     tmp.push_back(i - k);
24             std::fill(cnt.begin(), cnt.end(), 0);
25             for (int i = 0; i < n; ++i)

```

```

26         ++cnt[rk[i]];
27     for (int i = 1; i < n; ++i)
28         cnt[i] += cnt[i - 1];
29     for (int i = n - 1; i >= 0; --i)
30         sa[--cnt[rk[tmp[i]]]] = tmp[i];
31     std::swap(rk, tmp);
32     rk[sa[0]] = 0;
33     for (int i = 1; i < n; ++i)
34         rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i]
35             - 1] + k < tmp[sa[i] + k]);
36     k *= 2;
37 }
38 for (int i = 0, j = 0; i < n; ++i) {
39     if (rk[i] == 0) {
40         j = 0;
41     } else {
42         for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j];)
43             ++j;
44         lc[rk[i] - 1] = j;
45     }
46 }
47 };

```

6 Watashi 代码库 (备用)

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

Listing 42: rmq.cpp

```

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 };

```

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

Listing 43: lca.cpp

```

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;

```

6.3 树状数组

Listing 44: bit.cpp

```

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 };

```


6.4 并查集

Listing 45: union-find.cpp

```

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 };

```

6.5 轻重权树剖分

Listing 46: chain-decomp.cpp

```

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;

```

6.6 强连通分量

Listing 47: scc.cpp

```

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

```

```

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

```

6.7 双连通分量

Listing 48: bcc.cpp

```

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

```

```

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

```

```

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

```

6.8 二分图匹配

Listing 49: bimatch.cpp

```

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

```

```

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

```

6.9 最小费用最大流

Listing 50: flow.cpp

```

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

```

```

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

```

```

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

```

6.10 AhoCorasick 自动机

Listing 51: ac-automata.cpp

```

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

```



```

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

```

6.11 后缀数组

Listing 52: sa.cpp

```

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

```

```

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

```

6.12 LU 分解

Listing 53: lu.cpp

```

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[i][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     }
59 }
60
61 // LU(n - 1, a, r, c);
62 // solve(n - 1, a, r, c, b);

```

7 对一类问题的处理方法