

邪童の偷情笔记

这么菜



公司写题跟nm偷情一样

id: 856621855

偷情小分队在线打牛子

[-> 离开团队

邀请成员

成员



邪童 队长
华南农业大学

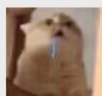


月仰
华南农业大学



oval_m
华南农业大学

公司写题跟nm偷情一样



哈哈哈哈哈你好他妈会讲话

精辟

能不能做队名

oval_m: 公司写题跟nm偷情一样

目录

jiangly 板子

- 一、杂类
 - 01 - int128 输出流自定义
 - 02 - 常用库函数重载
- 二、图与网络
 - 01 - 强连通分量缩点 (SCC)
 - 02 - 割边与割边缩点 (EBCC)
 - 03 - 二分图最大权匹配 (MaxAssignment 基于KM)
 - 04 - 一般图最大匹配 (Graph 带花树算法)
 - 05 - TwoSat (2-Sat)
 - 06A - 最大流 (Flow 旧版其一, 整数应用)
 - 06B - 最大流 (Flow 旧版其二, 浮点数应用)
 - 06C - 最大流 (MaxFlow 新版)
 - 07A - 费用流 (MCFGraph 最小费用可行流)
 - 07B - 费用流 (MCFGraph 最小费用最大流)
 - 08 - 树链剖分
- 三、树论、几何、多项式
 - 01 - 快速幂
 - 02 - 欧拉筛
 - 03 - 莫比乌斯函数筛 (莫比乌斯函数/反演)
 - 04 - 求解单个数的欧拉函数
 - 05 - 扩展欧几里得 (exGCD)
 - 06 - 组合数 (Comb+MInt & MLong)
 - 07 - 二项式 (Binomial 任意模数计算)
 - 08 - 素数测试与因数分解 (Miller-Rabin & Pollard-Rho)
 - 09 - 平面几何
 - 10 - 静态凸包
 - 11A - 多项式相关 (Poly 旧版)
 - 11B - 多项式相关 (Poly+MInt & MLong 新版)
- 四、数据结构
 - 01A - 树状数组 (Fenwick 旧版)
 - 01B - 树状数组 (Fenwick 新版)
 - 02 - 并查集 (DSU)
 - 03A - 线段树 (SegmentTree 基础区间加乘)
 - 03B - 线段树 (SegmentTree+Info 查找前驱后继)
 - 03C - 线段树 (SegmentTree+Info+Merge 区间合并)

- 04A - 懒标记线段树 (LazySegmentTree 基础区间修改)
- 04B - 懒标记线段树 (LazySegmentTree 查找前驱后继)
- 04C - 懒标记线段树 (LazySegmentTree 二分修改)
- 05A - 取模类 (MLong & MInt)
- 05B - 取模类 (MLong & MInt 新版)
- 06 - 状压RMQ (RMQ)
- 07 - Splay
- 08 - 其他平衡树
- 09 - 分数四则运算 (Frac)
- 10 - 线性基 (Basis)
- 五、字符串
 - 01 - 马拉车 (Manacher)
 - 02 - Z函数
 - 03 - 后缀数组 (SA)
 - 04A - 后缀自动机 (SuffixAutomaton 旧版)
 - 04B - 后缀自动机 (SAM 新版)
 - 05 - 回文自动机 (PAM)
 - 06A - AC自动机 (AC 旧版)
 - 06B - AC自动机 (AhoCorasick 新版)
 - 07 - 随机生成模底 字符串哈希 (例题)

其它

- BigInt
- 点分治
- 树分治
- 网络流封装
- bitset bfs
- 封装 multiset 对顶堆
- Miller-Rabin 大素数判定
- Pollard rho 大整数分解
- 双模哈希封装
- 哈希 + 数据结构
- 树与图上的计数问题
- SA 封装
- 排列组合公式
- Let it Rot

int128 输出流自定义

```
1  using i128 = __int128;
2
3  std::ostream &operator<<(std::ostream &os, i128 n) {
4      std::string s;
5      while (n) {
6          s += '0' + n % 10;
7          n /= 10;
8      }
9      std::reverse(s.begin(), s.end());
10     return os << s;
11 }
```

常用库函数重载

```
1  using i64 = long long;
2  using i128 = __int128;
3
4  i64 ceilDiv(i64 n, i64 m) {
5      if (n >= 0) {
6          return (n + m - 1) / m;
7      } else {
8          return n / m;
9      }
10 }
11
12 i64 floorDiv(i64 n, i64 m) {
13     if (n >= 0) {
14         return n / m;
15     } else {
16         return (n - m + 1) / m;
17     }
18 }
19
20 template<class T>
21 void chmax(T &a, T b) {
22     if (a < b) {
23         a = b;
24     }
25 }
26
27 i128 gcd(i128 a, i128 b) {
28     return b ? gcd(b, a % b) : a;
29 }
```

强连通分量缩点 (SCC)

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

```
53         if (dfn[i] == -1) {
54             dfs(i);
55         }
56     }
57     return bel;
58 }
59 };
```

割边与割边缩点 (EBCC)

```
1  std::set<std::pair<int, int>> E;
2
3  struct EBCC {
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     EBCC() {}
11     EBCC(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         adj[v].push_back(u);
28     }
29
30     void dfs(int x, int p) {
31         dfn[x] = low[x] = cur++;
32         stk.push_back(x);
33
34         for (auto y : adj[x]) {
35             if (y == p) {
36                 continue;
37             }
38             if (dfn[y] == -1) {
39                 E.emplace(x, y);
40                 dfs(y, x);
41                 low[x] = std::min(low[x], low[y]);
42             } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
43                 E.emplace(x, y);
44                 low[x] = std::min(low[x], dfn[y]);
45             }
46         }
47
48         if (dfn[x] == low[x]) {
49             int y;
50             do {
51                 y = stk.back();
52                 bel[y] = cnt;
```



```

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

```

二分图最大权匹配 (MaxAssignment 基于KM)

```
1  template<class T>
2  struct MaxAssignment {
3      public:
4          T solve(int nx, int ny, std::vector<std::vector<T>> a) {
5              assert(0 <= nx && nx <= ny);
6              assert(int(a.size()) == nx);
7              for (int i = 0; i < nx; ++i) {
8                  assert(int(a[i].size()) == ny);
9                  for (auto x : a[i])
10                     assert(x >= 0);
11              }
12
13              auto update = [&](int x) {
14                  for (int y = 0; y < ny; ++y) {
15                      if (lx[x] + ly[y] - a[x][y] < slack[y]) {
16                          slack[y] = lx[x] + ly[y] - a[x][y];
17                          slackx[y] = x;
18                      }
19                  }
20              };
21
22              costs.resize(nx + 1);
23              costs[0] = 0;
24              lx.assign(nx, std::numeric_limits<T>::max());
25              ly.assign(ny, 0);
26              xy.assign(nx, -1);
27              yx.assign(ny, -1);
28              slackx.resize(ny);
29              for (int cur = 0; cur < nx; ++cur) {
30                  std::queue<int> que;
31                  visx.assign(nx, false);
32                  visy.assign(ny, false);
33                  slack.assign(ny, std::numeric_limits<T>::max());
34                  p.assign(nx, -1);
35
36                  for (int x = 0; x < nx; ++x) {
37                      if (xy[x] == -1) {
38                          que.push(x);
39                          visx[x] = true;
40                          update(x);
41                      }
42                  }
43
44                  int ex, ey;
45                  bool found = false;
46                  while (!found) {
47                      while (!que.empty() && !found) {
48                          auto x = que.front();
49                          que.pop();
50                          for (int y = 0; y < ny; ++y) {
51                              if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
52                                  if (yx[y] == -1) {
```

```

53         ex = x;
54         ey = y;
55         found = true;
56         break;
57     }
58     que.push(yx[y]);
59     p[yx[y]] = x;
60     visy[y] = visx[yx[y]] = true;
61     update(yx[y]);
62     }
63     }
64 }
65 if (found)
66     break;
67
68 T delta = std::numeric_limits<T>::max();
69 for (int y = 0; y < ny; ++y)
70     if (!visy[y])
71         delta = std::min(delta, slack[y]);
72 for (int x = 0; x < nx; ++x)
73     if (visx[x])
74         lx[x] -= delta;
75 for (int y = 0; y < ny; ++y) {
76     if (visy[y]) {
77         ly[y] += delta;
78     } else {
79         slack[y] -= delta;
80     }
81 }
82 for (int y = 0; y < ny; ++y) {
83     if (!visy[y] && slack[y] == 0) {
84         if (yx[y] == -1) {
85             ex = slackx[y];
86             ey = y;
87             found = true;
88             break;
89         }
90         que.push(yx[y]);
91         p[yx[y]] = slackx[y];
92         visy[y] = visx[yx[y]] = true;
93         update(yx[y]);
94     }
95 }
96 }
97
98 costs[cur + 1] = costs[cur];
99 for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
100     costs[cur + 1] += a[x][y];
101     if (xy[x] != -1)
102         costs[cur + 1] -= a[x][xy[x]];
103     ty = xy[x];
104     xy[x] = y;
105     yx[y] = x;
106 }
107 }
108 return costs[nx];

```

```
109     }
110     std::vector<int> assignment() {
111         return xy;
112     }
113     std::pair<std::vector<T>, std::vector<T>> labels() {
114         return std::make_pair(lx, ly);
115     }
116     std::vector<T> weights() {
117         return costs;
118     }
119 private:
120     std::vector<T> lx, ly, slack, costs;
121     std::vector<int> xy, yx, p, slackx;
122     std::vector<bool> visx, visy;
123 };
```

一般图最大匹配 (Graph 带花树算法)

```
1 struct Graph {
2     int n;
3     std::vector<std::vector<int>> e;
4     Graph(int n) : n(n), e(n) {}
5     void addEdge(int u, int v) {
6         e[u].push_back(v);
7         e[v].push_back(u);
8     }
9     std::vector<int> findMatching() {
10         std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
11
12         // disjoint set union
13         auto find = [&](int u) {
14             while (f[u] != u)
15                 u = f[u] = f[f[u]];
16             return u;
17         };
18
19         auto lca = [&](int u, int v) {
20             u = find(u);
21             v = find(v);
22             while (u != v) {
23                 if (dep[u] < dep[v])
24                     std::swap(u, v);
25                 u = find(link[match[u]]);
26             }
27             return u;
28         };
29
30         std::queue<int> que;
31         auto blossom = [&](int u, int v, int p) {
32             while (find(u) != p) {
33                 link[u] = v;
34                 v = match[u];
35                 if (vis[v] == 0) {
36                     vis[v] = 1;
37                     que.push(v);
38                 }
39                 f[u] = f[v] = p;
40                 u = link[v];
41             }
42         };
43
44         // find an augmenting path starting from u and augment (if exist)
45         auto augment = [&](int u) {
46
47             while (!que.empty())
48                 que.pop();
49
50             std::iota(f.begin(), f.end(), 0);
51
```

```

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

TwoSat (2-Sat)

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


最大流 (Flow 旧版其一, 整数应用)

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

```
53     }
54     return f - r;
55 }
56 void addEdge(int u, int v, T c) {
57     g[u].push_back(e.size());
58     e.emplace_back(v, c);
59     g[v].push_back(e.size());
60     e.emplace_back(u, 0);
61 }
62 T maxFlow(int s, int t) {
63     T ans = 0;
64     while (bfs(s, t)) {
65         cur.assign(n, 0);
66         ans += dfs(s, t, std::numeric_limits<T>::max());
67     }
68     return ans;
69 }
70 };
```

最大流 (Flow 旧版其二, 浮点数应用)

```
1  template<class T>
2  struct Flow {
3      const int n;
4      struct Edge {
5          int to;
6          T cap;
7          Edge(int to, T cap) : to(to), cap(cap) {}
8      };
9      std::vector<Edge> e;
10     std::vector<std::vector<int>>> g;
11     std::vector<int> cur, h;
12     Flow(int n) : n(n), g(n) {}
13
14     bool bfs(int s, int t) {
15         h.assign(n, -1);
16         std::queue<int> que;
17         h[s] = 0;
18         que.push(s);
19         while (!que.empty()) {
20             const int u = que.front();
21             que.pop();
22             for (int i : g[u]) {
23                 auto [v, c] = e[i];
24                 if (c > 0 && h[v] == -1) {
25                     h[v] = h[u] + 1;
26                     if (v == t) {
27                         return true;
28                     }
29                     que.push(v);
30                 }
31             }
32         }
33         return false;
34     }
35
36     T dfs(int u, int t, T f) {
37         if (u == t) {
38             return f;
39         }
40         auto r = f;
41         double res = 0;
42         for (int &i = cur[u]; i < int(g[u].size()); ++i) {
43             const int j = g[u][i];
44             auto [v, c] = e[j];
45             if (c > 0 && h[v] == h[u] + 1) {
46                 auto a = dfs(v, t, std::min(r, c));
47                 res += a;
48                 e[j].cap -= a;
49                 e[j ^ 1].cap += a;
50                 r -= a;
51             }
52             if (r == 0) {
53                 return f;
54             }
55         }
56         cur[u] = i;
57         return res;
58     }
59
60     T maxflow(int s, int t) {
61         int flow = 0;
62         while (bfs(s, t)) {
63             flow += dfs(s, t, INF);
64         }
65         return flow;
66     }
67 }
```

```
53         }
54     }
55 }
56     return res;
57 }
58 void addEdge(int u, int v, T c) {
59     g[u].push_back(e.size());
60     e.emplace_back(v, c);
61     g[v].push_back(e.size());
62     e.emplace_back(u, 0);
63 }
64 T maxFlow(int s, int t) {
65     T ans = 0;
66     while (bfs(s, t)) {
67         cur.assign(n, 0);
68         ans += dfs(s, t, 1E100);
69     }
70     return ans;
71 }
72 };
```

最大流 (MaxFlow 新版)

```
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 };
```

费用流 (MCFGraph 最小费用可行流)

```
1 struct MCFGraph {
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,
15int>>, std::greater<std::pair<i64, int>>> que;
16        dis[s] = 0;
17        que.emplace(0, s);
18        while (!que.empty()) {
19            i64 d = que.top().first;
20            int u = que.top().second;
21            que.pop();
22            if (dis[u] < d) continue;
23            for (int i : g[u]) {
24                int v = e[i].v;
25                int c = e[i].c;
26                int f = e[i].f;
27                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
28                    dis[v] = d + h[u] - h[v] + f;
29                    pre[v] = i;
30                    que.emplace(dis[v], v);
31                }
32            }
33        }
34        return dis[t] != std::numeric_limits<i64>::max();
35    }
36    MCFGraph(int n) : n(n), g(n) {}
37    void addEdge(int u, int v, int c, int f) {
38        if (f < 0) {
39            g[u].push_back(e.size());
40            e.emplace_back(v, 0, f);
41            g[v].push_back(e.size());
42            e.emplace_back(u, c, -f);
43        } else {
44            g[u].push_back(e.size());
45            e.emplace_back(v, c, f);
46            g[v].push_back(e.size());
47            e.emplace_back(u, 0, -f);
48        }
49    }
50    std::pair<int, i64> flow(int s, int t) {
51        int flow = 0;
52        i64 cost = 0;
```



```

52     h.assign(n, 0);
53     while (dijkstra(s, t)) {
54         for (int i = 0; i < n; ++i) h[i] += dis[i];
55         int aug = std::numeric_limits<int>::max();
56         for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug,
e[pre[i]].c);
57         for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
58             e[pre[i]].c -= aug;
59             e[pre[i] ^ 1].c += aug;
60         }
61         flow += aug;
62         cost += i64(aug) * h[t];
63     }
64     return std::make_pair(flow, cost);
65 }
66 };

```

费用流 (MCFGraph 最小费用最大流)

```
1 struct MCFGraph {
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,
15int>>, std::greater<std::pair<i64, int>>> que;
16        dis[s] = 0;
17        que.emplace(0, s);
18        while (!que.empty()) {
19            i64 d = que.top().first;
20            int u = que.top().second;
21            que.pop();
22            if (dis[u] < d) continue;
23            for (int i : g[u]) {
24                int v = e[i].v;
25                int c = e[i].c;
26                int f = e[i].f;
27                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
28                    dis[v] = d + h[u] - h[v] + f;
29                    pre[v] = i;
30                    que.emplace(dis[v], v);
31                }
32            }
33        }
34        return dis[t] != std::numeric_limits<i64>::max();
35    }
36    MCFGraph(int n) : n(n), g(n) {}
37    void addEdge(int u, int v, int c, int f) {
38        g[u].push_back(e.size());
39        e.emplace_back(v, c, f);
40        g[v].push_back(e.size());
41        e.emplace_back(u, 0, -f);
42    }
43    std::pair<int, i64> flow(int s, int t) {
44        int flow = 0;
45        i64 cost = 0;
46        h.assign(n, 0);
47        while (dijkstra(s, t)) {
48            for (int i = 0; i < n; ++i) h[i] += dis[i];
49            int aug = std::numeric_limits<int>::max();
50            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug,
51e[pre[i]].c);
52            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
```

```
51         e[pre[i]].c -= aug;
52         e[pre[i] ^ 1].c += aug;
53     }
54     flow += aug;
55     cost += i64(aug) * h[t];
56 }
57 return std::make_pair(flow, cost);
58 }
59 };
```

树链剖分 (HLD)

```
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     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 };
```

快速幂

```
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 }
```

欧拉筛

```
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 }
```


莫比乌斯函数筛（莫比乌斯函数/反演）

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

```

53     return ans;
54 }
55
56 int main() {
57     std::ios::sync_with_stdio(false);
58     std::cin.tie(nullptr);
59
60     sieve(N);
61
62     int L, R;
63     std::cin >> L >> R;
64     L -= 1;
65
66     Z ans = 0;
67     for (int l = 1, r; l <= R; l = r + 1) {
68         r = R / (R / l);
69         if (l <= L) {
70             r = std::min(r, L / (L / l));
71         }
72
73         ans += (power(Z(2), R / l - L / l) - 1) * (sumMu(r) - sumMu(l - 1));
74     }
75
76     std::cout << ans << "\n";
77
78     return 0;
79 }

```

求解单个数的欧拉函数

```
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 }
```

扩展欧几里得 (exGCD)

```
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 }
```

组合数 (Comb+MInt & MLong)

```
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;
```

二项式 (Binomial 任意模数计算)

```
1  std::vector<std::pair<int, int>> factorize(int n) {
2      std::vector<std::pair<int, int>> factors;
3      for (int i = 2; static_cast<long long>(i) * i <= n; i++) {
4          if (n % i == 0) {
5              int t = 0;
6              for (; n % i == 0; n /= i)
7                  ++t;
8              factors.emplace_back(i, t);
9          }
10     }
11     if (n > 1)
12         factors.emplace_back(n, 1);
13     return factors;
14 }
15 constexpr int power(int base, i64 exp) {
16     int res = 1;
17     for (; exp > 0; base *= base, exp /= 2) {
18         if (exp % 2 == 1) {
19             res *= base;
20         }
21     }
22     return res;
23 }
24 constexpr int power(int base, i64 exp, int mod) {
25     int res = 1 % mod;
26     for (; exp > 0; base = 1LL * base * base % mod, exp /= 2) {
27         if (exp % 2 == 1) {
28             res = 1LL * res * base % mod;
29         }
30     }
31     return res;
32 }
33 int inverse(int a, int m) {
34     int g = m, r = a, x = 0, y = 1;
35     while (r != 0) {
36         int q = g / r;
37         g %= r;
38         std::swap(g, r);
39         x -= q * y;
40         std::swap(x, y);
41     }
42     return x < 0 ? x + m : x;
43 }
44 int solveModuloEquations(const std::vector<std::pair<int, int>> &e) {
45     int m = 1;
46     for (std::size_t i = 0; i < e.size(); i++) {
47         m *= e[i].first;
48     }
49     int res = 0;
50     for (std::size_t i = 0; i < e.size(); i++) {
51         int p = e[i].first;
52         res = (res + 1LL * e[i].second * (m / p) * inverse(m / p, p)) % m;
```

```

53     }
54     return res;
55 }
56 constexpr int N = 1E5;
57 class Binomial {
58     const int mod;
59 private:
60     const std::vector<std::pair<int, int>> factors;
61     std::vector<int> pk;
62     std::vector<std::vector<int>> prod;
63     static constexpr i64 exponent(i64 n, int p) {
64         i64 res = 0;
65         for (n /= p; n > 0; n /= p) {
66             res += n;
67         }
68         return res;
69     }
70     int product(i64 n, std::size_t i) {
71         int res = 1;
72         int p = factors[i].first;
73         for (; n > 0; n /= p) {
74             res = 1LL * res * power(prod[i].back(), n / pk[i], pk[i]) %
pk[i] * prod[i][n % pk[i]] % pk[i];
75         }
76         return res;
77     }
78 public:
79     Binomial(int mod) : mod(mod), factors(factorize(mod)) {
80         pk.resize(factors.size());
81         prod.resize(factors.size());
82         for (std::size_t i = 0; i < factors.size(); i++) {
83             int p = factors[i].first;
84             int k = factors[i].second;
85             pk[i] = power(p, k);
86             prod[i].resize(std::min(N + 1, pk[i]));
87             prod[i][0] = 1;
88             for (int j = 1; j < prod[i].size(); j++) {
89                 if (j % p == 0) {
90                     prod[i][j] = prod[i][j - 1];
91                 } else {
92                     prod[i][j] = 1LL * prod[i][j - 1] * j % pk[i];
93                 }
94             }
95         }
96     }
97     int operator()(i64 n, i64 m) {
98         if (n < m || m < 0) {
99             return 0;
100         }
101         std::vector<std::pair<int, int>> ans(factors.size());
102         for (int i = 0; i < factors.size(); i++) {
103             int p = factors[i].first;
104             int k = factors[i].second;
105             int e = exponent(n, p) - exponent(m, p) - exponent(n - m, p);
106             if (e >= k) {
107                 ans[i] = std::make_pair(pk[i], 0);

```

```
108         } else {
109             int pn = product(n, i);
110             int pm = product(m, i);
111             int pd = product(n - m, i);
112             int res = 1LL * pn * inverse(pm, pk[i]) % pk[i] *
inverse(pd, pk[i]) % pk[i] * power(p, e) % pk[i];
113             ans[i] = std::make_pair(pk[i], res);
114         }
115     }
116     return solveModuloEquations(ans);
117 }
118 };
```


素数测试与因式分解 (Miller-Rabin & Pollard-Rho)

```
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 }

```

平面几何

```
1  template<class T>
2  struct Point {
3      T x;
4      T y;
5      Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {}
6
7      template<class U>
8      operator Point<U>() {
9          return Point<U>(U(x), U(y));
10     }
11     Point &operator+=(Point p) & {
12         x += p.x;
13         y += p.y;
14         return *this;
15     }
16     Point &operator--=(Point p) & {
17         x -= p.x;
18         y -= p.y;
19         return *this;
20     }
21     Point &operator*=(T v) & {
22         x *= v;
23         y *= v;
24         return *this;
25     }
26     Point operator-() const {
27         return Point(-x, -y);
28     }
29     friend Point operator+(Point a, Point b) {
30         return a += b;
31     }
32     friend Point operator-(Point a, Point b) {
33         return a -= b;
34     }
35     friend Point operator*(Point a, T b) {
36         return a *= b;
37     }
38     friend Point operator*(T a, Point b) {
39         return b *= a;
40     }
41     friend bool operator==(Point a, Point b) {
42         return a.x == b.x && a.y == b.y;
43     }
44     friend std::istream &operator>>(std::istream &is, Point &p) {
45         return is >> p.x >> p.y;
46     }
47     friend std::ostream &operator<<(std::ostream &os, Point p) {
48         return os << "(" << p.x << ", " << p.y << ")";
49     }
50 };
51
52 template<class T>
```

```

53 T dot(Point<T> a, Point<T> b) {
54     return a.x * b.x + a.y * b.y;
55 }
56
57 template<class T>
58 T cross(Point<T> a, Point<T> b) {
59     return a.x * b.y - a.y * b.x;
60 }
61
62 template<class T>
63 T square(Point<T> p) {
64     return dot(p, p);
65 }
66
67 template<class T>
68 double length(Point<T> p) {
69     return std::sqrt(double(square(p)));
70 }
71
72 long double length(Point<long double> p) {
73     return std::sqrt(square(p));
74 }
75
76 template<class T>
77 struct Line {
78     Point<T> a;
79     Point<T> b;
80     Line(Point<T> a_ = Point<T>(), Point<T> b_ = Point<T>()) : a(a_), b(b_)
81     {}
82 };
83
84 template<class T>
85 Point<T> rotate(Point<T> a) {
86     return Point(-a.y, a.x);
87 }
88
89 template<class T>
90 int sgn(Point<T> a) {
91     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
92 }
93
94 template<class T>
95 bool pointOnLineLeft(Point<T> p, Line<T> l) {
96     return cross(l.b - l.a, p - l.a) > 0;
97 }
98
99 template<class T>
100 Point<T> lineIntersection(Line<T> l1, Line<T> l2) {
101     return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) /
102     cross(l2.b - l2.a, l1.a - l1.b));
103 }
104
105 template<class T>
106 bool pointOnSegment(Point<T> p, Line<T> l) {
107     return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x
108     && p.x <= std::max(l.a.x, l.b.x)

```

```

106     && std::min(l1.a.y, l1.b.y) <= p.y && p.y <= std::max(l1.a.y, l1.b.y);
107 }
108
109 template<class T>
110 bool pointInPolygon(Point<T> a, std::vector<Point<T>> p) {
111     int n = p.size();
112     for (int i = 0; i < n; i++) {
113         if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
114             return true;
115         }
116     }
117
118     int t = 0;
119     for (int i = 0; i < n; i++) {
120         auto u = p[i];
121         auto v = p[(i + 1) % n];
122         if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
123             t ^= 1;
124         }
125         if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
126             t ^= 1;
127         }
128     }
129
130     return t == 1;
131 }
132
133 // 0 : not intersect
134 // 1 : strictly intersect
135 // 2 : overlap
136 // 3 : intersect at endpoint
137 template<class T>
138 std::tuple<int, Point<T>, Point<T>> segmentIntersection(Line<T> l1, Line<T>
139 l2) {
140     if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
141         return {0, Point<T>(), Point<T>()};
142     }
143     if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
144         return {0, Point<T>(), Point<T>()};
145     }
146     if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
147         return {0, Point<T>(), Point<T>()};
148     }
149     if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
150         return {0, Point<T>(), Point<T>()};
151     }
152     if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
153         if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
154             return {0, Point<T>(), Point<T>()};
155         } else {
156             auto maxx1 = std::max(l1.a.x, l1.b.x);
157             auto minx1 = std::min(l1.a.x, l1.b.x);
158             auto maxy1 = std::max(l1.a.y, l1.b.y);
159             auto miny1 = std::min(l1.a.y, l1.b.y);
160             auto maxx2 = std::max(l2.a.x, l2.b.x);
161             auto minx2 = std::min(l2.a.x, l2.b.x);

```

```

161         auto maxy2 = std::max(l2.a.y, l2.b.y);
162         auto miny2 = std::min(l2.a.y, l2.b.y);
163         Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
164         Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
165         if (!pointOnSegment(p1, l1)) {
166             std::swap(p1.y, p2.y);
167         }
168         if (p1 == p2) {
169             return {3, p1, p2};
170         } else {
171             return {2, p1, p2};
172         }
173     }
174 }
175 auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
176 auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
177 auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
178 auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);
179
180 if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 >
0) || (cp3 < 0 && cp4 < 0)) {
181     return {0, Point<T>(), Point<T>()};
182 }
183
184 Point p = lineIntersection(l1, l2);
185 if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
186     return {1, p, p};
187 } else {
188     return {3, p, p};
189 }
190 }
191
192 template<class T>
193 bool segmentInPolygon(Line<T> l, std::vector<Point<T>> p) {
194     int n = p.size();
195     if (!pointInPolygon(l.a, p)) {
196         return false;
197     }
198     if (!pointInPolygon(l.b, p)) {
199         return false;
200     }
201     for (int i = 0; i < n; i++) {
202         auto u = p[i];
203         auto v = p[(i + 1) % n];
204         auto w = p[(i + 2) % n];
205         auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
206
207         if (t == 1) {
208             return false;
209         }
210         if (t == 0) {
211             continue;
212         }
213         if (t == 2) {
214             if (pointOnSegment(v, l) && v != l.a && v != l.b) {
215                 if (cross(v - u, w - v) > 0) {

```

```

216         return false;
217     }
218 }
219 } else {
220     if (p1 != u && p1 != v) {
221         if (pointOnLineLeft(l.a, Line(v, u))
222             || pointOnLineLeft(l.b, Line(v, u))) {
223             return false;
224         }
225     } else if (p1 == v) {
226         if (l.a == v) {
227             if (pointOnLineLeft(u, l)) {
228                 if (pointOnLineLeft(w, l)
229                     && pointOnLineLeft(w, Line(u, v))) {
230                     return false;
231                 }
232             } else {
233                 if (pointOnLineLeft(w, l)
234                     || pointOnLineLeft(w, Line(u, v))) {
235                     return false;
236                 }
237             }
238         } else if (l.b == v) {
239             if (pointOnLineLeft(u, Line(l.b, l.a))) {
240                 if (pointOnLineLeft(w, Line(l.b, l.a)
241                     && pointOnLineLeft(w, Line(u, v))) {
242                     return false;
243                 }
244             } else {
245                 if (pointOnLineLeft(w, Line(l.b, l.a)
246                     || pointOnLineLeft(w, Line(u, v))) {
247                     return false;
248                 }
249             }
250         } else {
251             if (pointOnLineLeft(u, l)) {
252                 if (pointOnLineLeft(w, Line(l.b, l.a)
253                     || pointOnLineLeft(w, Line(u, v))) {
254                     return false;
255                 }
256             } else {
257                 if (pointOnLineLeft(w, l)
258                     || pointOnLineLeft(w, Line(u, v))) {
259                     return false;
260                 }
261             }
262         }
263     }
264 }
265 }
266 return true;
267 }
268
269 template<class T>
270 std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
271     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {

```

```

272     auto d1 = l1.b - l1.a;
273     auto d2 = l2.b - l2.a;
274
275     if (sgn(d1) != sgn(d2)) {
276         return sgn(d1) == 1;
277     }
278
279     return cross(d1, d2) > 0;
280 });
281
282 std::deque<Line<T>> ls;
283 std::deque<Point<T>> ps;
284 for (auto l : lines) {
285     if (ls.empty()) {
286         ls.push_back(l);
287         continue;
288     }
289
290     while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
291         ps.pop_back();
292         ls.pop_back();
293     }
294
295     while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
296         ps.pop_front();
297         ls.pop_front();
298     }
299
300     if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
301         if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
302
303             if (!pointOnLineLeft(ls.back().a, l)) {
304                 assert(ls.size() == 1);
305                 ls[0] = l;
306             }
307             continue;
308         }
309         return {};
310     }
311
312     ps.push_back(lineIntersection(ls.back(), l));
313     ls.push_back(l);
314 }
315
316 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
317     ps.pop_back();
318     ls.pop_back();
319 }
320 if (ls.size() <= 2) {
321     return {};
322 }
323 ps.push_back(lineIntersection(ls[0], ls.back()));
324
325 return std::vector(ps.begin(), ps.end());
326 }

```


静态凸包

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

```

53     }
54
55     for (auto a : p) {
56         while (h.size() > 1 && cross(a - h.back(), a - h[h.size() - 2]) <=
0) {
57             h.pop_back();
58         }
59         while (l.size() > 1 && cross(a - l.back(), a - l[l.size() - 2]) >=
0) {
60             l.pop_back();
61         }
62         l.push_back(a);
63         h.push_back(a);
64     }
65
66     l.pop_back();
67     std::reverse(h.begin(), h.end());
68     h.pop_back();
69     l.insert(l.end(), h.begin(), h.end());
70     return l;
71 }

```

多项式相关 (Poly 旧版)

```
1  std::vector<int> rev;
2  std::vector<Z> roots{0, 1};
3  void dft(std::vector<Z> &a) {
4      int n = a.size();
5
6      if (int(rev.size()) != n) {
7          int k = __builtin_ctz(n) - 1;
8          rev.resize(n);
9          for (int i = 0; i < n; i++) {
10             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
11         }
12     }
13
14     for (int i = 0; i < n; i++) {
15         if (rev[i] < i) {
16             std::swap(a[i], a[rev[i]]);
17         }
18     }
19     if (int(roots.size()) < n) {
20         int k = __builtin_ctz(roots.size());
21         roots.resize(n);
22         while ((1 << k) < n) {
23             Z e = power(Z(3), (P - 1) >> (k + 1));
24             for (int i = 1 << (k - 1); i < (1 << k); i++) {
25                 roots[2 * i] = roots[i];
26                 roots[2 * i + 1] = roots[i] * e;
27             }
28             k++;
29         }
30     }
31     for (int k = 1; k < n; k *= 2) {
32         for (int i = 0; i < n; i += 2 * k) {
33             for (int j = 0; j < k; j++) {
34                 Z u = a[i + j];
35                 Z v = a[i + j + k] * roots[k + j];
36                 a[i + j] = u + v;
37                 a[i + j + k] = u - v;
38             }
39         }
40     }
41 }
42 void idft(std::vector<Z> &a) {
43     int n = a.size();
44     std::reverse(a.begin() + 1, a.end());
45     dft(a);
46     Z inv = (1 - P) / n;
47     for (int i = 0; i < n; i++) {
48         a[i] *= inv;
49     }
50 }
51 struct Poly {
52     std::vector<Z> a;
```

```

53     Poly() {}
54     explicit Poly(int size, std::function<Z(int)> f = [](int) { return 0;
    }) : a(size) {
55         for (int i = 0; i < size; i++) {
56             a[i] = f(i);
57         }
58     }
59     Poly(const std::vector<Z> &a) : a(a) {}
60     Poly(const std::initializer_list<Z> &a) : a(a) {}
61     int size() const {
62         return a.size();
63     }
64     void resize(int n) {
65         a.resize(n);
66     }
67     Z operator[](int idx) const {
68         if (idx < size()) {
69             return a[idx];
70         } else {
71             return 0;
72         }
73     }
74     Z &operator[](int idx) {
75         return a[idx];
76     }
77     Poly mulxk(int k) const {
78         auto b = a;
79         b.insert(b.begin(), k, 0);
80         return Poly(b);
81     }
82     Poly modxk(int k) const {
83         k = std::min(k, size());
84         return Poly(std::vector<Z>(a.begin(), a.begin() + k));
85     }
86     Poly divxk(int k) const {
87         if (size() <= k) {
88             return Poly();
89         }
90         return Poly(std::vector<Z>(a.begin() + k, a.end()));
91     }
92     friend Poly operator+(const Poly &a, const Poly &b) {
93         std::vector<Z> res(std::max(a.size(), b.size()));
94         for (int i = 0; i < int(res.size()); i++) {
95             res[i] = a[i] + b[i];
96         }
97         return Poly(res);
98     }
99     friend Poly operator-(const Poly &a, const Poly &b) {
100         std::vector<Z> res(std::max(a.size(), b.size()));
101         for (int i = 0; i < int(res.size()); i++) {
102             res[i] = a[i] - b[i];
103         }
104         return Poly(res);
105     }
106     friend Poly operator-(const Poly &a) {
107         std::vector<Z> res(a.size());

```

```

108     for (int i = 0; i < int(res.size()); i++) {
109         res[i] = -a[i];
110     }
111     return Poly(res);
112 }
113 friend Poly operator*(Poly a, Poly b) {
114     if (a.size() == 0 || b.size() == 0) {
115         return Poly();
116     }
117     if (a.size() < b.size()) {
118         std::swap(a, b);
119     }
120     if (b.size() < 128) {
121         Poly c(a.size() + b.size() - 1);
122         for (int i = 0; i < a.size(); i++) {
123             for (int j = 0; j < b.size(); j++) {
124                 c[i + j] += a[i] * b[j];
125             }
126         }
127         return c;
128     }
129     int sz = 1, tot = a.size() + b.size() - 1;
130     while (sz < tot) {
131         sz *= 2;
132     }
133     a.a.resize(sz);
134     b.a.resize(sz);
135     dft(a.a);
136     dft(b.a);
137     for (int i = 0; i < sz; ++i) {
138         a.a[i] = a[i] * b[i];
139     }
140     idft(a.a);
141     a.resize(tot);
142     return a;
143 }
144 friend Poly operator*(Z a, Poly b) {
145     for (int i = 0; i < int(b.size()); i++) {
146         b[i] *= a;
147     }
148     return b;
149 }
150 friend Poly operator*(Poly a, Z b) {
151     for (int i = 0; i < int(a.size()); i++) {
152         a[i] *= b;
153     }
154     return a;
155 }
156 Poly &operator+=(Poly b) {
157     return (*this) = (*this) + b;
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*=(Z b) {
166         return (*this) = (*this) * b;
167     }
168     Poly deriv() const {
169         if (a.empty()) {
170             return Poly();
171         }
172         std::vector<Z> res(size() - 1);
173         for (int i = 0; i < size() - 1; ++i) {
174             res[i] = (i + 1) * a[i + 1];
175         }
176         return Poly(res);
177     }
178     Poly integr() const {
179         std::vector<Z> res(size() + 1);
180         for (int i = 0; i < size(); ++i) {
181             res[i + 1] = a[i] / (i + 1);
182         }
183         return Poly(res);
184     }
185     Poly inv(int m) const {
186         Poly x{a[0].inv()};
187         int k = 1;
188         while (k < m) {
189             k *= 2;
190             x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
191         }
192         return x.modxk(m);
193     }
194     Poly log(int m) const {
195         return (deriv() * inv(m)).integr().modxk(m);
196     }
197     Poly exp(int m) const {
198         Poly x{1};
199         int k = 1;
200         while (k < m) {
201             k *= 2;
202             x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
203         }
204         return x.modxk(m);
205     }
206     Poly pow(int k, int m) const {
207         int i = 0;
208         while (i < size() && a[i].val() == 0) {
209             i++;
210         }
211         if (i == size() || 1LL * i * k >= m) {
212             return Poly(std::vector<Z>(m));
213         }
214         Z v = a[i];
215         auto f = divxk(i) * v.inv();
216         return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) *
power(v, k);
217     }
218     Poly sqrt(int m) const {

```

```

219     Poly x{1};
220     int k = 1;
221     while (k < m) {
222         k *= 2;
223         x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
224     }
225     return x.modxk(m);
226 }
227 Poly mult(Poly b) const {
228     if (b.size() == 0) {
229         return Poly();
230     }
231     int n = b.size();
232     std::reverse(b.a.begin(), b.a.end());
233     return ((*this) * b).divxk(n - 1);
234 }
235 std::vector<Z> eval(std::vector<Z> x) const {
236     if (size() == 0) {
237         return std::vector<Z>(x.size(), 0);
238     }
239     const int n = std::max(int(x.size()), size());
240     std::vector<Poly> q(4 * n);
241     std::vector<Z> ans(x.size());
242     x.resize(n);
243     std::function<void(int, int, int)> build = [&](int p, int l, int r)
244 {
245     if (r - l == 1) {
246         q[p] = Poly{1, -x[l]};
247     } else {
248         int m = (l + r) / 2;
249         build(2 * p, l, m);
250         build(2 * p + 1, m, r);
251         q[p] = q[2 * p] * q[2 * p + 1];
252     }
253 };
254 build(1, 0, n);
255 std::function<void(int, int, int, const Poly &)> work = [&](int p,
256 int l, int r, const Poly &num) {
257     if (r - l == 1) {
258         if (l < int(ans.size())) {
259             ans[l] = num[0];
260         }
261     } else {
262         int m = (l + r) / 2;
263         work(2 * p, l, m, num.mult(q[2 * p + 1]).modxk(m - 1));
264         work(2 * p + 1, m, r, num.mult(q[2 * p]).modxk(r - m));
265     }
266 };
267 work(1, 0, n, mult(q[1].inv(n)));
268 return ans;
269 }
270 };

```


多项式相关 (Poly+MInt & MLong 新版)

```
1  std::vector<int> rev;
2  template<int P>
3  std::vector<MInt<P>> roots{0, 1};
4
5  template<int P>
6  constexpr MInt<P> findPrimitiveRoot() {
7      MInt<P> i = 2;
8      int k = __builtin_ctz(P - 1);
9      while (true) {
10         if (power(i, (P - 1) / 2) != 1) {
11             break;
12         }
13         i += 1;
14     }
15     return power(i, (P - 1) >> k);
16 }
17
18 template<int P>
19 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
20
21 template<>
22 constexpr MInt<998244353> primitiveRoot<998244353> {31};
23
24 template<int P>
25 constexpr void dft(std::vector<MInt<P>> &a) {
26     int n = a.size();
27
28     if (int(rev.size()) != n) {
29         int k = __builtin_ctz(n) - 1;
30         rev.resize(n);
31         for (int i = 0; i < n; i++) {
32             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
33         }
34     }
35
36     for (int i = 0; i < n; i++) {
37         if (rev[i] < i) {
38             std::swap(a[i], a[rev[i]]);
39         }
40     }
41     if (roots<P>.size() < n) {
42         int k = __builtin_ctz(roots<P>.size());
43         roots<P>.resize(n);
44         while ((1 << k) < n) {
45             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k
- 1));
46             for (int i = 1 << (k - 1); i < (1 << k); i++) {
47                 roots<P>[2 * i] = roots<P>[i];
48                 roots<P>[2 * i + 1] = roots<P>[i] * e;
49             }
50             k++;
51         }
```

```

52     }
53     for (int k = 1; k < n; k *= 2) {
54         for (int i = 0; i < n; i += 2 * k) {
55             for (int j = 0; j < k; j++) {
56                 MInt<P> u = a[i + j];
57                 MInt<P> v = a[i + j + k] * roots<P>[k + j];
58                 a[i + j] = u + v;
59                 a[i + j + k] = u - v;
60             }
61         }
62     }
63 }
64
65 template<int P>
66 constexpr void idft(std::vector<MInt<P>> &a) {
67     int n = a.size();
68     std::reverse(a.begin() + 1, a.end());
69     dft(a);
70     MInt<P> inv = (1 - P) / n;
71     for (int i = 0; i < n; i++) {
72         a[i] *= inv;
73     }
74 }
75
76 template<int P = 998244353>
77 struct Poly : public std::vector<MInt<P>> {
78     using value = MInt<P>;
79
80     Poly() : std::vector<value>() {}
81     explicit constexpr Poly(int n) : std::vector<value>(n) {}
82
83     explicit constexpr Poly(const std::vector<value> &a) :
84     std::vector<value>(a) {}
85
86     constexpr Poly(const std::initializer_list<value> &a) :
87     std::vector<value>(a) {}
88
89     template<class InputIt, class = std::_RequireInputIter<InputIt>>
90     explicit constexpr Poly(InputIt first, InputIt last) :
91     std::vector<value>(first, last) {}
92
93     template<class F>
94     explicit constexpr Poly(int n, F f) : std::vector<value>(n) {
95         for (int i = 0; i < n; i++) {
96             (*this)[i] = f(i);
97         }
98     }
99
100     constexpr Poly shift(int k) const {
101         if (k >= 0) {
102             auto b = *this;
103             b.insert(b.begin(), k, 0);
104             return b;
105         } else if (this->size() <= -k) {
106             return Poly();
107         } else {
108             return Poly(this->begin() + (-k), this->end());
109         }
110     }

```

```

105     }
106 }
107 constexpr Poly trunc(int k) const {
108     Poly f = *this;
109     f.resize(k);
110     return f;
111 }
112 constexpr friend Poly operator+(const Poly &a, const Poly &b) {
113     Poly res(std::max(a.size(), b.size()));
114     for (int i = 0; i < a.size(); i++) {
115         res[i] += a[i];
116     }
117     for (int i = 0; i < b.size(); i++) {
118         res[i] += b[i];
119     }
120     return res;
121 }
122 constexpr friend Poly operator-(const Poly &a, const Poly &b) {
123     Poly res(std::max(a.size(), b.size()));
124     for (int i = 0; i < a.size(); i++) {
125         res[i] += a[i];
126     }
127     for (int i = 0; i < b.size(); i++) {
128         res[i] -= b[i];
129     }
130     return res;
131 }
132 constexpr friend Poly operator-(const Poly &a) {
133     std::vector<Value> res(a.size());
134     for (int i = 0; i < int(res.size()); i++) {
135         res[i] = -a[i];
136     }
137     return Poly(res);
138 }
139 constexpr friend Poly operator*(Poly a, Poly b) {
140     if (a.size() == 0 || b.size() == 0) {
141         return Poly();
142     }
143     if (a.size() < b.size()) {
144         std::swap(a, b);
145     }
146     int n = 1, tot = a.size() + b.size() - 1;
147     while (n < tot) {
148         n *= 2;
149     }
150     if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
151         Poly c(a.size() + b.size() - 1);
152         for (int i = 0; i < a.size(); i++) {
153             for (int j = 0; j < b.size(); j++) {
154                 c[i + j] += a[i] * b[j];
155             }
156         }
157         return c;
158     }
159     a.resize(n);
160     b.resize(n);

```

```

161     dft(a);
162     dft(b);
163     for (int i = 0; i < n; ++i) {
164         a[i] *= b[i];
165     }
166     idft(a);
167     a.resize(tot);
168     return a;
169 }
170 constexpr friend Poly operator*(Value a, Poly b) {
171     for (int i = 0; i < int(b.size()); i++) {
172         b[i] *= a;
173     }
174     return b;
175 }
176 constexpr friend Poly operator*(Poly a, Value b) {
177     for (int i = 0; i < int(a.size()); i++) {
178         a[i] *= b;
179     }
180     return a;
181 }
182 constexpr friend Poly operator/(Poly a, Value b) {
183     for (int i = 0; i < int(a.size()); i++) {
184         a[i] /= b;
185     }
186     return a;
187 }
188 constexpr Poly &operator+=(Poly b) {
189     return (*this) = (*this) + b;
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*=(Value b) {
198     return (*this) = (*this) * b;
199 }
200 constexpr Poly &operator/=(Value b) {
201     return (*this) = (*this) / b;
202 }
203 constexpr Poly deriv() const {
204     if (this->empty()) {
205         return Poly();
206     }
207     Poly res(this->size() - 1);
208     for (int i = 0; i < this->size() - 1; ++i) {
209         res[i] = (i + 1) * (*this)[i + 1];
210     }
211     return res;
212 }
213 constexpr Poly integr() const {
214     Poly res(this->size() + 1);
215     for (int i = 0; i < this->size(); ++i) {
216         res[i + 1] = (*this)[i] / (i + 1);

```

```

217     }
218     return res;
219 }
220 constexpr Poly inv(int m) const {
221     Poly x{(*this)[0].inv()};
222     int k = 1;
223     while (k < m) {
224         k *= 2;
225         x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
226     }
227     return x.trunc(m);
228 }
229 constexpr Poly log(int m) const {
230     return (deriv() * inv(m)).integr().trunc(m);
231 }
232 constexpr Poly exp(int m) const {
233     Poly x{1};
234     int k = 1;
235     while (k < m) {
236         k *= 2;
237         x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
238     }
239     return x.trunc(m);
240 }
241 constexpr Poly pow(int k, int m) const {
242     int i = 0;
243     while (i < this->size() && (*this)[i] == 0) {
244         i++;
245     }
246     if (i == this->size() || 1LL * i * k >= m) {
247         return Poly(m);
248     }
249     value v = (*this)[i];
250     auto f = shift(-i) * v.inv();
251     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) *
power(v, k);
252 }
253 constexpr Poly sqrt(int m) const {
254     Poly x{1};
255     int k = 1;
256     while (k < m) {
257         k *= 2;
258         x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
259     }
260     return x.trunc(m);
261 }
262 constexpr Poly mult(Poly b) const {
263     if (b.size() == 0) {
264         return Poly();
265     }
266     int n = b.size();
267     std::reverse(b.begin(), b.end());
268     return ((*this) * b).shift(-(n - 1));
269 }
270 constexpr std::vector<Value> eval(std::vector<Value> x) const {
271     if (this->size() == 0) {

```

```

272         return std::vector<Value>(x.size(), 0);
273     }
274     const int n = std::max(x.size(), this->size());
275     std::vector<Poly> q(4 * n);
276     std::vector<Value> ans(x.size());
277     x.resize(n);
278     std::function<void(int, int, int)> build = [&](int p, int l, int r)
{
279         if (r - l == 1) {
280             q[p] = Poly{1, -x[l]};
281         } else {
282             int m = (l + r) / 2;
283             build(2 * p, l, m);
284             build(2 * p + 1, m, r);
285             q[p] = q[2 * p] * q[2 * p + 1];
286         }
287     };
288     build(1, 0, n);
289     std::function<void(int, int, int, const Poly &)> work = [&](int p,
int l, int r, const Poly &num) {
290         if (r - l == 1) {
291             if (l < int(ans.size())) {
292                 ans[l] = num[0];
293             }
294         } else {
295             int m = (l + r) / 2;
296             work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - 1));
297             work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
298         }
299     };
300     work(1, 0, n, mulT(q[1].inv(n)));
301     return ans;
302 }
303 };
304
305 template<int P = 998244353>
306 Poly<P> berlekampMassey(const Poly<P> &s) {
307     Poly<P> c;
308     Poly<P> oldC;
309     int f = -1;
310     for (int i = 0; i < s.size(); i++) {
311         auto delta = s[i];
312         for (int j = 1; j <= c.size(); j++) {
313             delta -= c[j - 1] * s[i - j];
314         }
315         if (delta == 0) {
316             continue;
317         }
318         if (f == -1) {
319             c.resize(i + 1);
320             f = i;
321         } else {
322             auto d = oldC;
323             d *= -1;
324             d.insert(d.begin(), 1);
325             MInt<P> df1 = 0;

```

```

326         for (int j = 1; j <= d.size(); j++) {
327             df1 += d[j - 1] * s[f + 1 - j];
328         }
329         assert(df1 != 0);
330         auto coef = delta / df1;
331         d *= coef;
332         Poly<P> zeros(i - f - 1);
333         zeros.insert(zeros.end(), d.begin(), d.end());
334         d = zeros;
335         auto temp = c;
336         c += d;
337         if (i - temp.size() > f - oldC.size()) {
338             oldC = temp;
339             f = i;
340         }
341     }
342 }
343 c *= -1;
344 c.insert(c.begin(), 1);
345 return c;
346 }
347
348
349 template<int P = 998244353>
350 MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
351     int m = q.size() - 1;
352     while (n > 0) {
353         auto newq = q;
354         for (int i = 1; i <= m; i += 2) {
355             newq[i] *= -1;
356         }
357         auto newp = p * newq;
358         newq = q * newq;
359         for (int i = 0; i < m; i++) {
360             p[i] = newp[i * 2 + n % 2];
361         }
362         for (int i = 0; i <= m; i++) {
363             q[i] = newq[i * 2];
364         }
365         n /= 2;
366     }
367     return p[0] / q[0];
368 }
369
370 struct Comb {
371     int n;
372     std::vector<Z> _fac;
373     std::vector<Z> _invfac;
374     std::vector<Z> _inv;
375
376     Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
377     Comb(int n) : Comb() {
378         init(n);
379     }
380
381     void init(int m) {

```

```

382     m = std::min(m, Z::getMod() - 1);
383     if (m <= n) return;
384     _fac.resize(m + 1);
385     _invfac.resize(m + 1);
386     _inv.resize(m + 1);
387
388     for (int i = n + 1; i <= m; i++) {
389         _fac[i] = _fac[i - 1] * i;
390     }
391     _invfac[m] = _fac[m].inv();
392     for (int i = m; i > n; i--) {
393         _invfac[i - 1] = _invfac[i] * i;
394         _inv[i] = _invfac[i] * _fac[i - 1];
395     }
396     n = m;
397 }
398
399 Z fac(int m) {
400     if (m > n) init(2 * m);
401     return _fac[m];
402 }
403
404 Z invfac(int m) {
405     if (m > n) init(2 * m);
406     return _invfac[m];
407 }
408
409 Z inv(int m) {
410     if (m > n) init(2 * m);
411     return _inv[m];
412 }
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++) {

```



```
438     pw[i] = pw[i - 1] * (m / 2);
439 }
440 for (int i = 0; i <= n; i++) {
441     pw[i] *= comb.invfac(i);
442 }
443 fm = fm.mult(pw);
444 for (int i = 0; i <= n; i++) {
445     fm[i] *= comb.invfac(i);
446 }
447 return f + fm;
448 }
```

树状数组 (Fenwick 旧版)

```
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         this->n = n;
12         a.assign(n, T());
13     }
14
15     void add(int x, T v) {
16         for (int i = x + 1; i <= n; i += i & -i) {
17             a[i - 1] += v;
18         }
19     }
20
21     T sum(int x) {
22         auto ans = T();
23         for (int i = x; i > 0; i -= i & -i) {
24             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 kth(T k) {
34         int x = 0;
35         for (int i = 1 << std::__lg(n); i; i /= 2) {
36             if (x + i <= n && k >= a[x + i - 1]) {
37                 x += i;
38                 k -= a[x - 1];
39             }
40         }
41         return x;
42     }
43 };
```

树状数组 (Fenwick 新版)

```
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 };
```

并查集 (DSU)

```
1 struct DSU {
2     std::vector<int> f, siz;
3
4     DSU() {}
5     DSU(int n) {
6         init(n);
7     }
8
9     void init(int n) {
10         f.resize(n);
11         std::iota(f.begin(), f.end(), 0);
12         siz.assign(n, 1);
13     }
14
15     int find(int x) {
16         while (x != f[x]) {
17             x = f[x] = f[f[x]];
18         }
19         return x;
20     }
21
22     bool same(int x, int y) {
23         return find(x) == find(y);
24     }
25
26     bool merge(int x, int y) {
27         x = find(x);
28         y = find(y);
29         if (x == y) {
30             return false;
31         }
32         siz[x] += siz[y];
33         f[y] = x;
34         return true;
35     }
36
37     int size(int x) {
38         return siz[find(x)];
39     }
40 };
```

线段树 (SegmentTree 基础区间加乘)

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

```
52     rangeMul(1, 0, n, x, y, v);
53 }
54
55 void add(int p, int l, int r, int x, int v) {
56     if (r - l == 1) {
57         sum[p] = (sum[p] + v) % P;
58         return;
59     }
60     int m = (l + r) / 2;
61     push(p);
62     if (x < m) {
63         add(2 * p, l, m, x, v);
64     } else {
65         add(2 * p + 1, m, r, x, v);
66     }
67     pull(p);
68 }
69
70 void add(int x, int v) {
71     add(1, 0, n, x, v);
72 }
73 };
```

线段树 (SegmentTree+Info 查找前驱后继)

```
1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5      SegmentTree() : n(0) {}
6      SegmentTree(int n_, Info v_ = Info()) {
7          init(n_, v_);
8      }
9      template<class T>
10     SegmentTree(std::vector<T> init_) {
11         init(init_);
12     }
13     void init(int n_, Info v_ = Info()) {
14         init(std::vector(n_, v_));
15     }
16     template<class T>
17     void init(std::vector<T> init_) {
18         n = init_.size();
19         info.assign(4 << std::__lg(n), Info());
20         std::function<void(int, int, int)> build = [&](int p, int l, int r)
21     {
22         if (r - l == 1) {
23             info[p] = init_[l];
24             return;
25         }
26         int m = (l + r) / 2;
27         build(2 * p, l, m);
28         build(2 * p + 1, m, r);
29         pull(p);
30     };
31     build(1, 0, n);
32     void pull(int p) {
33         info[p] = info[2 * p] + info[2 * p + 1];
34     }
35     void modify(int p, int l, int r, int x, const Info &v) {
36         if (r - l == 1) {
37             info[p] = v;
38             return;
39         }
40         int m = (l + r) / 2;
41         if (x < m) {
42             modify(2 * p, l, m, x, v);
43         } else {
44             modify(2 * p + 1, m, r, x, v);
45         }
46         pull(p);
47     }
48     void modify(int p, const Info &v) {
49         modify(1, 0, n, p, v);
50     }
51     Info rangeQuery(int p, int l, int r, int x, int y) {
```

```

52         if (l >= y || r <= x) {
53             return Info();
54         }
55         if (l >= x && r <= y) {
56             return info[p];
57         }
58         int m = (l + r) / 2;
59         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r,
x, y);
60     }
61     Info rangeQuery(int l, int r) {
62         return rangeQuery(1, 0, n, l, r);
63     }
64     template<class F>
65     int findFirst(int p, int l, int r, int x, int y, F pred) {
66         if (l >= y || r <= x || !pred(info[p])) {
67             return -1;
68         }
69         if (r - l == 1) {
70             return l;
71         }
72         int m = (l + r) / 2;
73         int res = findFirst(2 * p, l, m, x, y, pred);
74         if (res == -1) {
75             res = findFirst(2 * p + 1, m, r, x, y, pred);
76         }
77         return res;
78     }
79     template<class F>
80     int findFirst(int l, int r, F pred) {
81         return findFirst(1, 0, n, l, r, pred);
82     }
83     template<class F>
84     int findLast(int p, int l, int r, int x, int y, F pred) {
85         if (l >= y || r <= x || !pred(info[p])) {
86             return -1;
87         }
88         if (r - l == 1) {
89             return l;
90         }
91         int m = (l + r) / 2;
92         int res = findLast(2 * p + 1, m, r, x, y, pred);
93         if (res == -1) {
94             res = findLast(2 * p, l, m, x, y, pred);
95         }
96         return res;
97     }
98     template<class F>
99     int findLast(int l, int r, F pred) {
100         return findLast(1, 0, n, l, r, pred);
101     }
102 };
103 struct Info {
104     int cnt = 0;
105     i64 sum = 0;
106     i64 ans = 0;

```



```
107 };
108 Info operator+(Info a, Info b) {
109     Info c;
110     c.cnt = a.cnt + b.cnt;
111     c.sum = a.sum + b.sum;
112     c.ans = a.ans + b.ans + a.cnt * b.sum - a.sum * b.cnt;
113     return c;
114 }
```

线段树 (SegmentTree+Info+Merge 区间合并)

```
1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5      SegmentTree() : n(0) {}
6      SegmentTree(int n_, Info v_ = Info()) {
7          init(n_, v_);
8      }
9      template<class T>
10     SegmentTree(std::vector<T> init_) {
11         init(init_);
12     }
13     void init(int n_, Info v_ = Info()) {
14         init(std::vector(n_, v_));
15     }
16     template<class T>
17     void init(std::vector<T> init_) {
18         n = init_.size();
19         info.assign(4 << std::__lg(n), Info());
20         std::function<void(int, int, int)> build = [&](int p, int l, int r)
21     {
22         if (r - l == 1) {
23             info[p] = init_[l];
24             return;
25         }
26         int m = (l + r) / 2;
27         build(2 * p, l, m);
28         build(2 * p + 1, m, r);
29         pull(p);
30     };
31     build(1, 0, n);
32     void pull(int p) {
33         info[p] = info[2 * p] + info[2 * p + 1];
34     }
35     void modify(int p, int l, int r, int x, const Info &v) {
36         if (r - l == 1) {
37             info[p] = v;
38             return;
39         }
40         int m = (l + r) / 2;
41         if (x < m) {
42             modify(2 * p, l, m, x, v);
43         } else {
44             modify(2 * p + 1, m, r, x, v);
45         }
46         pull(p);
47     }
48     void modify(int p, const Info &v) {
49         modify(1, 0, n, p, v);
50     }
51     Info rangeQuery(int p, int l, int r, int x, int y) {
```

```

52         if (l >= y || r <= x) {
53             return Info();
54         }
55         if (l >= x && r <= y) {
56             return info[p];
57         }
58         int m = (l + r) / 2;
59         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r,
x, y);
60     }
61     Info rangeQuery(int l, int r) {
62         return rangeQuery(1, 0, n, l, r);
63     }
64     template<class F>
65     int findFirst(int p, int l, int r, int x, int y, F pred) {
66         if (l >= y || r <= x || !pred(info[p])) {
67             return -1;
68         }
69         if (r - l == 1) {
70             return l;
71         }
72         int m = (l + r) / 2;
73         int res = findFirst(2 * p, l, m, x, y, pred);
74         if (res == -1) {
75             res = findFirst(2 * p + 1, m, r, x, y, pred);
76         }
77         return res;
78     }
79     template<class F>
80     int findFirst(int l, int r, F pred) {
81         return findFirst(1, 0, n, l, r, pred);
82     }
83     template<class F>
84     int findLast(int p, int l, int r, int x, int y, F pred) {
85         if (l >= y || r <= x || !pred(info[p])) {
86             return -1;
87         }
88         if (r - l == 1) {
89             return l;
90         }
91         int m = (l + r) / 2;
92         int res = findLast(2 * p + 1, m, r, x, y, pred);
93         if (res == -1) {
94             res = findLast(2 * p, l, m, x, y, pred);
95         }
96         return res;
97     }
98     template<class F>
99     int findLast(int l, int r, F pred) {
100         return findLast(1, 0, n, l, r, pred);
101     }
102 };
103
104 struct Info {
105     int x = 0;
106     int cnt = 0;

```

```
107 };
108
109 Info operator+(Info a, Info b) {
110     if (a.x == b.x) {
111         return {a.x, a.cnt + b.cnt};
112     } else if (a.cnt > b.cnt) {
113         return {a.x, a.cnt - b.cnt};
114     } else {
115         return {b.x, b.cnt - a.cnt};
116     }
117 }
```

懒标记线段树 (LazySegmentTree 基础区间修改)

```
1  template<class Info, class Tag>
2  struct LazySegmentTree {
3      const int n;
4      std::vector<Info> info;
5      std::vector<Tag> tag;
6      LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 <<
std::__lg(n)) {}
7      LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size())
{
8          std::function<void(int, int, int)> build = [&](int p, int l, int r)
{
9              if (r - l == 1) {
10                 info[p] = init[l];
11                 return;
12             }
13             int m = (l + r) / 2;
14             build(2 * p, l, m);
15             build(2 * p + 1, m, r);
16             pull(p);
17         };
18         build(1, 0, n);
19     }
20     void pull(int p) {
21         info[p] = info[2 * p] + info[2 * p + 1];
22     }
23     void apply(int p, const Tag &v) {
24         info[p].apply(v);
25         tag[p].apply(v);
26     }
27     void push(int p) {
28         apply(2 * p, tag[p]);
29         apply(2 * p + 1, tag[p]);
30         tag[p] = Tag();
31     }
32     void modify(int p, int l, int r, int x, const Info &v) {
33         if (r - l == 1) {
34             info[p] = v;
35             return;
36         }
37         int m = (l + r) / 2;
38         push(p);
39         if (x < m) {
40             modify(2 * p, l, m, x, v);
41         } else {
42             modify(2 * p + 1, m, r, x, v);
43         }
44         pull(p);
45     }
46     void modify(int p, const Info &v) {
47         modify(1, 0, n, p, v);
48     }
49     Info rangeQuery(int p, int l, int r, int x, int y) {
```

```

50     if (l >= y || r <= x) {
51         return Info();
52     }
53     if (l >= x && r <= y) {
54         return info[p];
55     }
56     int m = (l + r) / 2;
57     push(p);
58     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r,
x, y);
59 }
60 Info rangeQuery(int l, int r) {
61     return rangeQuery(1, 0, n, l, r);
62 }
63 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
64     if (l >= y || r <= x) {
65         return;
66     }
67     if (l >= x && r <= y) {
68         apply(p, v);
69         return;
70     }
71     int m = (l + r) / 2;
72     push(p);
73     rangeApply(2 * p, l, m, x, y, v);
74     rangeApply(2 * p + 1, m, r, x, y, v);
75     pull(p);
76 }
77 void rangeApply(int l, int r, const Tag &v) {
78     return rangeApply(1, 0, n, l, r, v);
79 }
80 void half(int p, int l, int r) {
81     if (info[p].act == 0) {
82         return;
83     }
84     if ((info[p].min + 1) / 2 == (info[p].max + 1) / 2) {
85         apply(p, {-(info[p].min + 1) / 2});
86         return;
87     }
88     int m = (l + r) / 2;
89     push(p);
90     half(2 * p, l, m);
91     half(2 * p + 1, m, r);
92     pull(p);
93 }
94 void half() {
95     half(1, 0, n);
96 }
97 };
98
99 constexpr i64 inf = 1E18;
100
101 struct Tag {
102     i64 add = 0;
103
104     void apply(Tag t) {

```

```
105         add += t.add;
106     }
107 };
108
109 struct Info {
110     i64 min = inf;
111     i64 max = -inf;
112     i64 sum = 0;
113     i64 act = 0;
114
115     void apply(Tag t) {
116         min += t.add;
117         max += t.add;
118         sum += act * t.add;
119     }
120 };
121
122 Info operator+(Info a, Info b) {
123     Info c;
124     c.min = std::min(a.min, b.min);
125     c.max = std::max(a.max, b.max);
126     c.sum = a.sum + b.sum;
127     c.act = a.act + b.act;
128     return c;
129 }
```

懒标记线段树 (LazySegmentTree 查找前驱后继)

```
1  template<class Info, class Tag>
2  struct LazySegmentTree {
3      int n;
4      std::vector<Info> info;
5      std::vector<Tag> tag;
6      LazySegmentTree() : n(0) {}
7      LazySegmentTree(int n_, Info v_ = Info()) {
8          init(n_, v_);
9      }
10     template<class T>
11     LazySegmentTree(std::vector<T> init_) {
12         init(init_);
13     }
14     void init(int n_, Info v_ = Info()) {
15         init(std::vector(n_, v_));
16     }
17     template<class T>
18     void init(std::vector<T> init_) {
19         n = init_.size();
20         info.assign(4 << std::__lg(n), Info());
21         tag.assign(4 << std::__lg(n), Tag());
22         std::function<void(int, int, int)> build = [&](int p, int l, int r)
23         {
24             if (r - l == 1) {
25                 info[p] = init_[l];
26                 return;
27             }
28             int m = (l + r) / 2;
29             build(2 * p, l, m);
30             build(2 * p + 1, m, r);
31             pull(p);
32         };
33         build(1, 0, n);
34     void pull(int p) {
35         info[p] = info[2 * p] + info[2 * p + 1];
36     }
37     void apply(int p, const Tag &v) {
38         info[p].apply(v);
39         tag[p].apply(v);
40     }
41     void push(int p) {
42         apply(2 * p, tag[p]);
43         apply(2 * p + 1, tag[p]);
44         tag[p] = Tag();
45     }
46     void modify(int p, int l, int r, int x, const Info &v) {
47         if (r - l == 1) {
48             info[p] = v;
49             return;
50         }
51         int m = (l + r) / 2;
```



```

52     push(p);
53     if (x < m) {
54         modify(2 * p, l, m, x, v);
55     } else {
56         modify(2 * p + 1, m, r, x, v);
57     }
58     pull(p);
59 }
60 void modify(int p, const Info &v) {
61     modify(1, 0, n, p, v);
62 }
63 Info rangeQuery(int p, int l, int r, int x, int y) {
64     if (l >= y || r <= x) {
65         return Info();
66     }
67     if (l >= x && r <= y) {
68         return info[p];
69     }
70     int m = (l + r) / 2;
71     push(p);
72     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r,
x, y);
73 }
74 Info rangeQuery(int l, int r) {
75     return rangeQuery(1, 0, n, l, r);
76 }
77 void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
78     if (l >= y || r <= x) {
79         return;
80     }
81     if (l >= x && r <= y) {
82         apply(p, v);
83         return;
84     }
85     int m = (l + r) / 2;
86     push(p);
87     rangeApply(2 * p, l, m, x, y, v);
88     rangeApply(2 * p + 1, m, r, x, y, v);
89     pull(p);
90 }
91 void rangeApply(int l, int r, const Tag &v) {
92     return rangeApply(1, 0, n, l, r, v);
93 }
94 template<class F>
95 int findFirst(int p, int l, int r, int x, int y, F pred) {
96     if (l >= y || r <= x || !pred(info[p])) {
97         return -1;
98     }
99     if (r - l == 1) {
100         return l;
101     }
102     int m = (l + r) / 2;
103     push(p);
104     int res = findFirst(2 * p, l, m, x, y, pred);
105     if (res == -1) {
106         res = findFirst(2 * p + 1, m, r, x, y, pred);

```

```

107     }
108     return res;
109 }
110 template<class F>
111 int findFirst(int l, int r, F pred) {
112     return findFirst(1, 0, n, l, r, pred);
113 }
114 template<class F>
115 int findLast(int p, int l, int r, int x, int y, F pred) {
116     if (l >= y || r <= x || !pred(info[p])) {
117         return -1;
118     }
119     if (r - l == 1) {
120         return l;
121     }
122     int m = (l + r) / 2;
123     push(p);
124     int res = findLast(2 * p + 1, m, r, x, y, pred);
125     if (res == -1) {
126         res = findLast(2 * p, l, m, x, y, pred);
127     }
128     return res;
129 }
130 template<class F>
131 int findLast(int l, int r, F pred) {
132     return findLast(1, 0, n, l, r, pred);
133 }
134 };
135
136 struct Tag {
137     i64 a = 0, b = 0;
138     void apply(Tag t) {
139         a = std::min(a, b + t.a);
140         b += t.b;
141     }
142 };
143
144 int k;
145
146 struct Info {
147     i64 x = 0;
148     void apply(Tag t) {
149         x += t.a;
150         if (x < 0) {
151             x = (x % k + k) % k;
152         }
153         x += t.b - t.a;
154     }
155 };
156 Info operator+(Info a, Info b) {
157     return {a.x + b.x};
158 }

```

懒标记线段树 (LazySegmentTree 二分修改)

```
1  constexpr int inf = 1E9 + 1;
2  template<class Info, class Tag>
3  struct LazySegmentTree {
4      const int n;
5      std::vector<Info> info;
6      std::vector<Tag> tag;
7      LazySegmentTree(int n) : n(n), info(4 << std::__lg(n)), tag(4 <<
std::__lg(n)) {}
8      LazySegmentTree(std::vector<Info> init) : LazySegmentTree(init.size())
{
9          std::function<void(int, int, int)> build = [&](int p, int l, int r)
{
10             if (r - l == 1) {
11                 info[p] = init[l];
12                 return;
13             }
14             int m = (l + r) / 2;
15             build(2 * p, l, m);
16             build(2 * p + 1, m, r);
17             pull(p);
18         };
19         build(1, 0, n);
20     }
21     void pull(int p) {
22         info[p] = info[2 * p] + info[2 * p + 1];
23     }
24     void apply(int p, const Tag &v) {
25         info[p].apply(v);
26         tag[p].apply(v);
27     }
28     void push(int p) {
29         apply(2 * p, tag[p]);
30         apply(2 * p + 1, tag[p]);
31         tag[p] = Tag();
32     }
33     void modify(int p, int l, int r, int x, const Info &v) {
34         if (r - l == 1) {
35             info[p] = v;
36             return;
37         }
38         int m = (l + r) / 2;
39         push(p);
40         if (x < m) {
41             modify(2 * p, l, m, x, v);
42         } else {
43             modify(2 * p + 1, m, r, x, v);
44         }
45         pull(p);
46     }
47     void modify(int p, const Info &v) {
48         modify(1, 0, n, p, v);
49     }
```

```

50     Info rangeQuery(int p, int l, int r, int x, int y) {
51         if (l >= y || r <= x) {
52             return Info();
53         }
54         if (l >= x && r <= y) {
55             return info[p];
56         }
57         int m = (l + r) / 2;
58         push(p);
59         return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r,
x, y);
60     }
61     Info rangeQuery(int l, int r) {
62         return rangeQuery(1, 0, n, l, r);
63     }
64     void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
65         if (l >= y || r <= x) {
66             return;
67         }
68         if (l >= x && r <= y) {
69             apply(p, v);
70             return;
71         }
72         int m = (l + r) / 2;
73         push(p);
74         rangeApply(2 * p, l, m, x, y, v);
75         rangeApply(2 * p + 1, m, r, x, y, v);
76         pull(p);
77     }
78     void rangeApply(int l, int r, const Tag &v) {
79         return rangeApply(1, 0, n, l, r, v);
80     }
81     void maintainL(int p, int l, int r, int pre) {
82         if (info[p].difl > 0 && info[p].maxlowl < pre) {
83             return;
84         }
85         if (r - l == 1) {
86             info[p].max = info[p].maxlowl;
87             info[p].maxl = info[p].maxr = l;
88             info[p].maxlowl = info[p].maxlowr = -inf;
89             return;
90         }
91         int m = (l + r) / 2;
92         push(p);
93         maintainL(2 * p, l, m, pre);
94         pre = std::max(pre, info[2 * p].max);
95         maintainL(2 * p + 1, m, r, pre);
96         pull(p);
97     }
98     void maintainL() {
99         maintainL(1, 0, n, -1);
100     }
101     void maintainR(int p, int l, int r, int suf) {
102         if (info[p].difr > 0 && info[p].maxlowr < suf) {
103             return;
104         }

```

```

105         if (r - l == 1) {
106             info[p].max = info[p].maxlowl;
107             info[p].maxl = info[p].maxr = l;
108             info[p].maxlowl = info[p].maxlowr = -inf;
109             return;
110         }
111         int m = (l + r) / 2;
112         push(p);
113         maintainR(2 * p + 1, m, r, suf);
114         suf = std::max(suf, info[2 * p + 1].max);
115         maintainR(2 * p, l, m, suf);
116         pull(p);
117     }
118     void maintainR() {
119         maintainR(1, 0, n, -1);
120     }
121 };
122
123 struct Tag {
124     int add = 0;
125
126     void apply(Tag t) & {
127         add += t.add;
128     }
129 };
130
131 struct Info {
132     int max = -1;
133     int maxl = -1;
134     int maxr = -1;
135     int difl = inf;
136     int difr = inf;
137     int maxlowl = -inf;
138     int maxlowr = -inf;
139
140     void apply(Tag t) & {
141         if (max != -1) {
142             max += t.add;
143         }
144         difl += t.add;
145         difr += t.add;
146     }
147 };
148
149 Info operator+(Info a, Info b) {
150     Info c;
151     if (a.max > b.max) {
152         c.max = a.max;
153         c.maxl = a.maxl;
154         c.maxr = a.maxr;
155     } else if (a.max < b.max) {
156         c.max = b.max;
157         c.maxl = b.maxl;
158         c.maxr = b.maxr;
159     } else {
160         c.max = a.max;

```

```
161         c.maxl = a.maxl;
162         c.maxr = b.maxr;
163     }
164
165     c.difl = std::min(a.difl, b.difl);
166     c.difr = std::min(a.difr, b.difr);
167     if (a.max != -1) {
168         c.difl = std::min(c.difl, a.max - b.maxlowl);
169     }
170     if (b.max != -1) {
171         c.difr = std::min(c.difr, b.max - a.maxlowr);
172     }
173
174     if (a.max == -1) {
175         c.maxlowl = std::max(a.maxlowl, b.maxlowl);
176     } else {
177         c.maxlowl = a.maxlowl;
178     }
179     if (b.max == -1) {
180         c.maxlowr = std::max(a.maxlowr, b.maxlowr);
181     } else {
182         c.maxlowr = b.maxlowr;
183     }
184     return c;
185 }
```

取模类 (MLong & MInt)

```
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 };
```


取模类 (MLong & MInt 新版)

根据输入内容动态修改 MOD 的方法: `Z::setMod(p)`

```
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
103    &a) {
104        return os << a.val();
105    }
106    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
199 &a) {
200         return os << a.val();
201     }
202     friend constexpr bool operator==(MInt lhs, MInt rhs) {
203         return lhs.val() == rhs.val();
204     }
205     friend constexpr bool operator!=(MInt lhs, MInt rhs) {
206         return lhs.val() != rhs.val();
207     }
208 };
209
210 template<>
211 int MInt<0>::Mod = 998244353;
212
213 template<int V, int P>
214 constexpr MInt<P> CInv = MInt<P>(V).inv();
215
216 constexpr int P = 1000000007;
217 using Z = MInt<P>;

```


状压RMQ (RMQ)

```
1  template<class T,
2      class Cmp = std::less<T>>
3  struct RMQ {
4      const Cmp cmp = Cmp();
5      static constexpr unsigned B = 64;
6      using u64 = unsigned long long;
7      int n;
8      std::vector<std::vector<T>> a;
9      std::vector<T> pre, suf, ini;
10     std::vector<u64> stk;
11     RMQ() {}
12     RMQ(const std::vector<T> &v) {
13         init(v);
14     }
15     void init(const std::vector<T> &v) {
16         n = v.size();
17         pre = suf = ini = v;
18         stk.resize(n);
19         if (!n) {
20             return;
21         }
22         const int M = (n - 1) / B + 1;
23         const int lg = std::__lg(M);
24         a.assign(lg + 1, std::vector<T>(M));
25         for (int i = 0; i < M; i++) {
26             a[0][i] = v[i * B];
27             for (int j = 1; j < B && i * B + j < n; j++) {
28                 a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
29             }
30         }
31         for (int i = 1; i < n; i++) {
32             if (i % B) {
33                 pre[i] = std::min(pre[i], pre[i - 1], cmp);
34             }
35         }
36         for (int i = n - 2; i >= 0; i--) {
37             if (i % B != B - 1) {
38                 suf[i] = std::min(suf[i], suf[i + 1], cmp);
39             }
40         }
41         for (int j = 0; j < lg; j++) {
42             for (int i = 0; i + (2 << j) <= M; i++) {
43                 a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
44             }
45         }
46         for (int i = 0; i < M; i++) {
47             const int l = i * B;
48             const int r = std::min(1U * n, l + B);
49             u64 s = 0;
50             for (int j = 1; j < r; j++) {
51                 while (s && cmp(v[j], v[std::__lg(s) + 1])) {
52                     s ^= 1ULL << std::__lg(s);
```

```

53         }
54         s |= 1ULL << (j - 1);
55         stk[j] = s;
56     }
57 }
58 }
59 T operator()(int l, int r) {
60     if (l / B != (r - 1) / B) {
61         T ans = std::min(suf[l], pre[r - 1], cmp);
62         l = l / B + 1;
63         r = r / B;
64         if (l < r) {
65             int k = std::__lg(r - l);
66             ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
67         }
68         return ans;
69     } else {
70         int x = B * (l / B);
71         return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + 1];
72     }
73 }
74 };

```

Splay

```
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 }

```

```

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
22 : 0);
23 }
24 bool isroot(Node *t) {
25     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);
26 }
27 int pos(Node *t) {
28     return t->p->ch[1] == t;
29 }
30 void pushAll(Node *t) {
31     if (!isroot(t)) {
32         pushAll(t->p);
33     }
34     push(t);
35 }
36 void rotate(Node *t) {
37     Node *q = t->p;
38     int x = !pos(t);
39     q->ch[!x] = t->ch[x];
40     if (t->ch[x]) {
41         t->ch[x]->p = q;
42     }
43     t->p = q->p;
44     if (!isroot(q)) {
45         q->p->ch[pos(q)] = t;
46     }
47     t->ch[x] = q;
48     q->p = t;
49     pull(q);
50 }
51 void splay(Node *t) {
52     pushAll(t);
53     while (!isroot(t)) {
54         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 }

```

```

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 }
49 void pull(Node *t) {
50     t->prd = (t->ch[0] ? t->ch[0]->prd : Matrix()) * t->mat * (t->ch[1] ?
t->ch[1]->prd : Matrix());
51 }
52 bool isroot(Node *t) {
53     return t->p == nullptr || (t->p->ch[0] != t && t->p->ch[1] != t);

```

```

54 }
55 int pos(Node *t) {
56     return t->p->ch[1] == t;
57 }
58 void pushAll(Node *t) {
59     if (!isroot(t)) {
60         pushAll(t->p);
61     }
62     push(t);
63 }
64 void rotate(Node *t) {
65     Node *q = t->p;
66     int x = !pos(t);
67     q->ch[!x] = t->ch[x];
68     if (t->ch[x]) {
69         t->ch[x]->p = q;
70     }
71     t->p = q->p;
72     if (!isroot(q)) {
73         q->p->ch[pos(q)] = t;
74     }
75     t->ch[x] = q;
76     q->p = t;
77     pull(q);
78 }
79 void splay(Node *t) {
80     pushAll(t);
81     while (!isroot(t)) {
82         if (!isroot(t->p)) {
83             if (pos(t) == pos(t->p)) {
84                 rotate(t->p);
85             } else {
86                 rotate(t);
87             }
88         }
89         rotate(t);
90     }
91     pull(t);
92 }
93
94 std::array<i64, 4> get(Node *t) {
95     std::array<i64, 4> ans;
96     ans.fill(inf);
97     ans[3] = 0;
98     for (int i = 0; i < 3; i++) {
99         for (int j = 0; j < 4; j++) {
100             ans[i] = std::min(ans[i], t->prd[i][j]);
101         }
102     }
103     return ans;
104 }
105
106 void access(Node *t) {
107     std::array<i64, 4> old{};
108     for (Node *i = t, *q = nullptr; i; q = i, i = i->p) {
109         splay(i);

```

```
110     if (i->ch[1]) {
111         auto res = get(i->ch[1]);
112         i->sumg += res[0];
113         i->sumh += std::min({res[1], res[2], res[3]});
114         i->sumb += std::min({res[0], res[1], res[2], res[3]});
115     }
116     i->ch[1] = q;
117     i->sumg -= old[0];
118     i->sumh -= std::min({old[1], old[2], old[3]});
119     i->sumb -= std::min({old[0], old[1], old[2], old[3]});
120     old = get(i);
121     i->update();
122     pull(i);
123 }
124 splay(t);
125 }
```


其他平衡树

```
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 }
```

```

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 }

```

```

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 }
```

```

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 }

```


分数四则运算 (Frac)

```
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 };

```

线性基 (Basis)

```
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 };
```

马拉车 (Manacher)

```
1  std::vector<int> manacher(std::string s) {
2      std::string t = "#";
3      for (auto c : s) {
4          t += c;
5          t += '#';
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     {
15         r[i] += 1;
16     }
17     if (i + r[i] > j + r[j]) {
18         j = i;
19     }
20 }
21 return r;
22 }
```

Z函数

```
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 }
```

后缀数组 (SA)

```
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] <
11        s[b];});
12        rk[sa[0]] = 0;
13        for (int i = 1; i < n; ++i)
14            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
15        int k = 1;
16        std::vector<int> tmp, cnt(n);
17        tmp.reserve(n);
18        while (rk[sa[n - 1]] < n - 1) {
19            tmp.clear();
20            for (int i = 0; i < k; ++i)
21                tmp.push_back(n - k + i);
22            for (auto i : sa)
23                if (i >= k)
24                    tmp.push_back(i - k);
25            std::fill(cnt.begin(), cnt.end(), 0);
26            for (int i = 0; i < n; ++i)
27                ++cnt[rk[i]];
28            for (int i = 1; i < n; ++i)
29                cnt[i] += cnt[i - 1];
30            for (int i = n - 1; i >= 0; --i)
31                sa[--cnt[rk[tmp[i]]]] = tmp[i];
32            std::swap(rk, tmp);
33            rk[sa[0]] = 0;
34            for (int i = 1; i < n; ++i)
35                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] ||
36                sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
37            k *= 2;
38        }
39        for (int i = 0, j = 0; i < n; ++i) {
40            if (rk[i] == 0) {
41                j = 0;
42            } else {
43                for (j -= j > 0; i + j < n && sa[rk[i] - 1] + j < n && s[i +
44                j] == s[sa[rk[i] - 1] + j]; )
45                    ++j;
46                lc[rk[i] - 1] = j;
47            }
48        }
49    }
50};
```

后缀自动机 (SuffixAutomaton 旧版)

```
1 struct SuffixAutomaton {
2     static constexpr int ALPHABET_SIZE = 26, N = 5e5;
3     struct Node {
4         int len;
5         int link;
6         int next[ALPHABET_SIZE];
7         Node() : len(0), link(0), next{} {}
8     } t[2 * N];
9     int cntNodes;
10    SuffixAutomaton() {
11        cntNodes = 1;
12        std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
13        t[0].len = -1;
14    }
15    int extend(int p, int c) {
16        if (t[p].next[c]) {
17            int q = t[p].next[c];
18            if (t[q].len == t[p].len + 1)
19                return q;
20            int r = ++cntNodes;
21            t[r].len = t[p].len + 1;
22            t[r].link = t[q].link;
23            std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
24            t[q].link = r;
25            while (t[p].next[c] == q) {
26                t[p].next[c] = r;
27                p = t[p].link;
28            }
29            return r;
30        }
31        int cur = ++cntNodes;
32        t[cur].len = t[p].len + 1;
33        while (!t[p].next[c]) {
34            t[p].next[c] = cur;
35            p = t[p].link;
36        }
37        t[cur].link = extend(p, c);
38        return cur;
39    }
40};
```

后缀自动机 (SAM 新版)

```
1 struct SAM {
2     static constexpr int ALPHABET_SIZE = 26;
3     struct Node {
4         int len;
5         int link;
6         std::array<int, ALPHABET_SIZE> next;
7         Node() : len{}, link{}, next{} {}
8     };
9     std::vector<Node> t;
10    SAM() {
11        init();
12    }
13    void init() {
14        t.assign(2, Node());
15        t[0].next.fill(1);
16        t[0].len = -1;
17    }
18    int newNode() {
19        t.emplace_back();
20        return t.size() - 1;
21    }
22    int extend(int p, int c) {
23        if (t[p].next[c]) {
24            int q = t[p].next[c];
25            if (t[q].len == t[p].len + 1) {
26                return q;
27            }
28            int r = newNode();
29            t[r].len = t[p].len + 1;
30            t[r].link = t[q].link;
31            t[r].next = t[q].next;
32            t[q].link = r;
33            while (t[p].next[c] == q) {
34                t[p].next[c] = r;
35                p = t[p].link;
36            }
37            return r;
38        }
39        int cur = newNode();
40        t[cur].len = t[p].len + 1;
41        while (!t[p].next[c]) {
42            t[p].next[c] = cur;
43            p = t[p].link;
44        }
45        t[cur].link = extend(p, c);
46        return cur;
47    }
48    int extend(int p, char c, char offset = 'a') {
49        return extend(p, c - offset);
50    }
51
52    int next(int p, int x) {
```



```
53         return t[p].next[x];
54     }
55
56     int next(int p, char c, char offset = 'a') {
57         return next(p, c - 'a');
58     }
59
60     int link(int p) {
61         return t[p].link;
62     }
63
64     int len(int p) {
65         return t[p].len;
66     }
67
68     int size() {
69         return t.size();
70     }
71 };
```

回文自动机 (PAM)

```
1  struct PAM {
2      static constexpr int ALPHABET_SIZE = 28;
3      struct Node {
4          int len;
5          int link;
6          int cnt;
7          std::array<int, ALPHABET_SIZE> next;
8          Node() : len{}, link{}, cnt{}, next{} {}
9      };
10     std::vector<Node> t;
11     int suff;
12     std::string s;
13     PAM() {
14         init();
15     }
16     void init() {
17         t.assign(2, Node());
18         t[0].len = -1;
19         suff = 1;
20         s.clear();
21     }
22     int newNode() {
23         t.emplace_back();
24         return t.size() - 1;
25     }
26
27     bool add(char c, char offset = 'a') {
28         int pos = s.size();
29         s += c;
30         int let = c - offset;
31         int cur = suff, curlen = 0;
32
33         while (true) {
34             curlen = t[cur].len;
35             if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
36                 break;
37             cur = t[cur].link;
38         }
39         if (t[cur].next[let]) {
40             suff = t[cur].next[let];
41             return false;
42         }
43
44         int num = newNode();
45         suff = num;
46         t[num].len = t[cur].len + 2;
47         t[cur].next[let] = num;
48
49         if (t[num].len == 1) {
50             t[num].link = 1;
51             t[num].cnt = 1;
52             return true;
53         }
```

```
53     }
54
55     while (true) {
56         cur = t[cur].link;
57         curlen = t[cur].len;
58         if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
59             t[num].link = t[cur].next[let];
60             break;
61         }
62     }
63
64     t[num].cnt = 1 + t[t[num].link].cnt;
65
66     return true;
67 }
68 };
69
70 PAM pam;
```

AC自动机 (AC 旧版)

```
1  constexpr int N = 3e5 + 30, A = 26;
2
3  struct Node {
4      int fail;
5      int sum;
6      int next[A];
7      Node() : fail(-1), sum(0) {
8          std::memset(next, -1, sizeof(next));
9      }
10 } node[N];
11
12 int cnt = 0;
13 int bin[N];
14 int nBin = 0;
15
16 int newNode() {
17     int p = nBin > 0 ? bin[--nBin] : cnt++;
18     node[p] = Node();
19     return p;
20 }
21
22 struct AC {
23     std::vector<int> x;
24     AC(AC &&a) : x(std::move(a.x)) {}
25     AC(std::vector<std::string> s, std::vector<int> w) {
26         x = {newNode(), newNode()};
27         std::fill(node[x[0]].next, node[x[0]].next + A, x[1]);
28         node[x[1]].fail = x[0];
29
30         for (int i = 0; i < int(s.size()); i++) {
31             int p = x[1];
32             for (int j = 0; j < int(s[i].length()); j++) {
33                 int c = s[i][j] - 'a';
34                 if (node[p].next[c] == -1) {
35                     int u = newNode();
36                     x.push_back(u);
37                     node[p].next[c] = u;
38                 }
39                 p = node[p].next[c];
40             }
41             node[p].sum += w[i];
42         }
43
44         std::queue<int> que;
45         que.push(x[1]);
46         while (!que.empty()) {
47             int u = que.front();
48             que.pop();
49             node[u].sum += node[node[u].fail].sum;
50             for (int c = 0; c < A; c++) {
51                 if (node[u].next[c] == -1) {
52                     node[u].next[c] = node[node[u].fail].next[c];
```

```

53         } else {
54             node[node[u].next[c]].fail = node[node[u].fail].next[c];
55             que.push(node[u].next[c]);
56         }
57     }
58 }
59 }
60 ~AC() {
61     for (auto p : x) {
62         bin[nBin++] = p;
63     }
64 }
65 i64 query(const std::string &s) const {
66     i64 ans = 0;
67     int p = x[1];
68     for (int i = 0; i < int(s.length()); i++) {
69         int c = s[i] - 'a';
70         p = node[p].next[c];
71         ans += node[p].sum;
72     }
73     return ans;
74 }
75 };

```

AC自动机 (AhoCorasick 新版)

```
1 struct AhoCorasick {
2     static constexpr int ALPHABET = 26;
3     struct Node {
4         int len;
5         int link;
6         std::array<int, ALPHABET> next;
7         Node() : link{}, next{} {}
8     };
9
10    std::vector<Node> t;
11
12    AhoCorasick() {
13        init();
14    }
15
16    void init() {
17        t.assign(2, Node());
18        t[0].next.fill(1);
19        t[0].len = -1;
20    }
21
22    int newNode() {
23        t.emplace_back();
24        return t.size() - 1;
25    }
26
27    int add(const std::vector<int> &a) {
28        int p = 1;
29        for (auto x : a) {
30            if (t[p].next[x] == 0) {
31                t[p].next[x] = newNode();
32                t[t[p].next[x]].len = t[p].len + 1;
33            }
34            p = t[p].next[x];
35        }
36        return p;
37    }
38
39    int add(const std::string &a, char offset = 'a') {
40        std::vector<int> b(a.size());
41        for (int i = 0; i < a.size(); i++) {
42            b[i] = a[i] - offset;
43        }
44        return add(b);
45    }
46
47    void work() {
48        std::queue<int> q;
49        q.push(1);
50
51        while (!q.empty()) {
52            int x = q.front();
```

```

53         q.pop();
54
55         for (int i = 0; i < ALPHABET; i++) {
56             if (t[x].next[i] == 0) {
57                 t[x].next[i] = t[t[x].link].next[i];
58             } else {
59                 t[t[x].next[i]].link = t[t[x].link].next[i];
60                 q.push(t[x].next[i]);
61             }
62         }
63     }
64 }
65
66 int next(int p, int x) {
67     return t[p].next[x];
68 }
69
70 int next(int p, char c, char offset = 'a') {
71     return next(p, c - 'a');
72 }
73
74 int link(int p) {
75     return t[p].link;
76 }
77
78 int len(int p) {
79     return t[p].len;
80 }
81
82 int size() {
83     return t.size();
84 }
85 };

```

随机生成模底 字符串哈希（例题）

```
1  #include <bits/stdc++.h>
2
3  using i64 = long long;
4
5  bool isprime(int n) {
6      if (n <= 1) {
7          return false;
8      }
9      for (int i = 2; i * i <= n; i++) {
10         if (n % i == 0) {
11             return false;
12         }
13     }
14     return true;
15 }
16
17 int findPrime(int n) {
18     while (!isprime(n)) {
19         n++;
20     }
21     return n;
22 }
23
24 using Hash = std::array<int, 2>;
25
26 int main() {
27     std::ios::sync_with_stdio(false);
28     std::cin.tie(nullptr);
29
30     std::mt19937
rng(std::chrono::steady_clock::now().time_since_epoch().count());
31
32     const int P = findPrime(rng() % 900000000 + 100000000);
33
34     std::string s, x;
35     std::cin >> s >> x;
36
37     int n = s.length();
38     int m = x.length();
39
40     std::vector<int> h(n + 1), p(n + 1);
41     for (int i = 0; i < n; i++) {
42         h[i + 1] = (10LL * h[i] + s[i] - '0') % P;
43     }
44     p[0] = 1;
45     for (int i = 0; i < n; i++) {
46         p[i + 1] = 10LL * p[i] % P;
47     }
48
49     auto get = [&](int l, int r) {
50         return (h[r] + 1LL * (P - h[l]) * p[r - l]) % P;
51     };
```



```

52
53     int px = 0;
54     for (auto c : x) {
55         px = (10LL * px + c - '0') % P;
56     }
57
58     for (int i = 0; i <= n - 2 * (m - 1); i++) {
59         if ((get(i, i + m - 1) + get(i + m - 1, i + 2 * m - 2)) % P == px)
60     {
61         std::cout << i + 1 << " " << i + m - 1 << "\n";
62         std::cout << i + m << " " << i + 2 * m - 2 << "\n";
63         return 0;
64     }
65
66     std::vector<int> z(m + 1), f(n + 1);
67     z[0] = m;
68
69     for (int i = 1, j = -1; i < m; i++) {
70         if (j != -1) {
71             z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
72         }
73         while (z[i] + i < m && x[z[i]] == x[z[i] + i]) {
74             z[i]++;
75         }
76         if (j == -1 || i + z[i] > j + z[j]) {
77             j = i;
78         }
79     }
80     for (int i = 0, j = -1; i < n; i++) {
81         if (j != -1) {
82             f[i] = std::max(0, std::min(j + f[j] - i, z[i - j]));
83         }
84         while (f[i] + i < n && f[i] < m && x[f[i]] == s[f[i] + i]) {
85             f[i]++;
86         }
87         if (j == -1 || i + f[i] > j + f[j]) {
88             j = i;
89         }
90     }
91
92     for (int i = 0; i + m <= n; i++) {
93         int l = std::min(m, f[i]);
94
95         for (auto j : { m - l, m - l - 1 }) {
96             if (j <= 0) {
97                 continue;
98             }
99             if (j <= i && (get(i - j, i) + get(i, i + m)) % P == px) {
100                 std::cout << i - j + 1 << " " << i << "\n";
101                 std::cout << i + 1 << " " << i + m << "\n";
102                 return 0;
103             }
104             if (i + m + j <= n && (get(i, i + m) + get(i + m, i + m + j)) %
P == px) {
105                 std::cout << i + 1 << " " << i + m << "\n";

```

```
106         std::cout << i + m + 1 << " " << i + m + j << "\n";
107         return 0;
108     }
109 }
110 }
111
112 return 0;
113 }
```

BigInt

```
1  #ifndef UseFFT
2  /***** FFT 板子 *****/
3  namespace FFT{
4  // 精度保证: 系数总和小于 1e15 开longdouble
5  // 精度够(maybe总和小于1e10)可以不开longdouble
6  // 先调用fft_init()
7  // FFT_MAXN = 2^k
8  // fft_init() to precalc FFT_MAXN-th roots
9  typedef long double db;
10 const int FFT_MX=2097152, N =4001000;
11 const db pi=acosl(-1.);
12 struct cp{
13     db a,b;
14     cp operator+(const cp&y)const{return (cp){a+y.a,b+y.b}};
15     cp operator-(const cp&y)const{return (cp){a-y.a,b-y.b}};
16     cp operator*(const cp&y)const{return (cp){a*y.a-b*y.b,a*y.b+b*y.a}};
17     cp operator!()const{return (cp){a,-b}};
18 }nw[FFT_MX+1];
19 int bitrev[FFT_MX];
20
21 void dft(cp*a,int n,int flag=1){
22     int d=0;
23     while((1<<d)*n!=FFT_MX) d++;
24     For(i,0,n-1) if(i < (bitrev[i]>>d))
25         swap(a[i], a[bitrev[i]>>d]);
26     for (int l=2;l<=n;l<=1){
27         int del=FFT_MX/l*flag;
28         for (int i=0;i<n;i+=l){
29             cp *le=a+i, *ri=a+i+(l>>1), *w=(flag==1) ? nw : nw+FFT_MX;
30             For(k,0,(l>>1)-1){
31                 cp ne=*ri**w;
32                 *ri=*le-ne, *le=*le+ne;
33                 le++, ri++, w+=del;
34             }
35         }
36     }
37     if(flag!=1) For(i,0,n-1)
38         a[i].a/=n, a[i].b/=n;
39 }
40 void fft_init(){
41     int L=0;
42     while((1<<L)!=FFT_MX) L++;
43     bitrev[0]=0;
44     For(i,1,FFT_MX-1)
45         bitrev[i] = bitrev[i>>1]>>1 | ((i&1)<<(L-1));
46     nw[0]=nw[FFT_MX]=(cp){1,0};
47     For(i,0,FFT_MX)nw[i] = (cp){cosl(2*pi/FFT_MX*i), sinl(2*pi/FFT_MX*i)};
48 //very slow
49 }
50 // n, m 分别为a, b的最高次幂, 数组a的范围为[0, n], b为[0, m], c转整数四舍五入
51 void polymul(db *a, int n, db *b, int m, db *c) {
52     static cp f[FFT_MX>>1], g[FFT_MX>>1], t[FFT_MX>>1];
```

```

52     int N=2;
53     while(N<=n+m)N<=1;
54     For(i,0,N-1)    // 此N非全局的N
55         if(i&1){
56             f[i>>1].b=(i<=n)?a[i]:0.0;
57             g[i>>1].b=(i<=m)?b[i]:0.0;
58         }else{
59             f[i>>1].a=(i<=n)?a[i]:0.0;
60             g[i>>1].a=(i<=m)?b[i]:0.0;
61         }
62     dft(f,N>>1); dft(g,N>>1);
63     int del=FFT_MX/(N>>1);
64     cp qua=(cp){0,0.25}, one=(cp){1,0}, four=(cp){4,0}, *w=nw;
65     For(i,0,(N>>1)-1){
66         int j=i?(N>>1)-i:0;
67         t[i] = (four*!(f[j]*g[j])-(!f[j]-f[i])*(!g[j]-g[i])*(one+*w))*qua;
68         w+=del;
69     }
70     dft(t,N>>1,-111);
71     For(i,0,n+m)c[i]=(i&1) ? t[i>>1].a : t[i>>1].b;
72 }
73 // 这实际抄的时候可以和上面结合一下 少个复制的常数
74 db A[N], B[N], C[N];
75 vector<ll> vectorMul(const vector<ll>& a, const vector<ll>& b){
76     int n = a.size(), m = b.size();
77     if(n==0 || m==0)return {};
78     n--, m--;
79     For(i,0,n)A[i] = a[i];
80     For(i,0,m)B[i] = b[i];
81     polymul(A, n, B, m, C);
82     vector<ll> res(n+m+1);
83     For(i,0,n+m) res[i] = C[i]+0.5;
84     return res;
85 }
86 }
87 #endif

```

```

1 namespace BigIntSP{
2 // 十进制高精度板子，将 D 个十进制位压到一起
3 // FFT 时最好<=5
4 const int D = 1;
5 const int B = pow(10, D);
6 struct BigInt
7 {
8     // ===== 初始化部分 =====
9     // int, string 转 BigInt; BigInt 转 string 输出；取绝对值，取反；高位推进
10
11     int sign = 0;
12     vector<ll> v;
13     BigInt(ll x = 0){
14         set(x);
15     }
16     BigInt(const string &s) {
17         set(s);
18     }
19     void set(ll x){
20         v.clear();
21         sign = 0;
22         if (x < 0) x *= -1, sign = 1;
23         while (x) {
24             v.push_back(x % B);
25             x /= B;
26         }
27     }
28     void set(const string& s){
29         sign = 0;
30         v.clear();
31         int beg = 0;
32         if (s[0] == '-')
33             beg++, sign = 1;
34         int add = 0, cnt = 0, base = 1;
35         for (int i = s.size()-1; i >= beg; i--) {
36             if (cnt == D) {
37                 v.push_back(add);
38                 cnt = add = 0;
39                 base = 1;
40             }
41             add = (s[i] - '0') * base + add;
42             cnt++;
43             base *= 10;
44         }
45         if (add) v.push_back(add);
46     }
47     BigInt operator-() const {
48         BigInt res = *this;
49         res.sign ^= 1;
50         return res;
51     }
52     BigInt abs() const {
53         BigInt res = *this;
54         res.sign = 0;

```

```

55     return res;
56 }
57 ll& operator[](const int i) { return v[i]; }
58 int size() const { return v.size(); }
59 void norm() { // 向高位推一遍进位
60     For(i,0,(ll)v.size()-2) {
61         if (v[i] >= 0) {
62             v[i + 1] += v[i] / B;
63             v[i] %= B;
64         } else {
65             int c = (-v[i] + B - 1) / B;
66             v[i] += c * B;
67             v[i + 1] -= c;
68         }
69     }
70     while (!v.empty() && v.back() >= B) {
71         int c = v.back() / B;
72         v.back() %= B;
73         v.push_back(c);
74     }
75     while (!v.empty() && v.back() == 0) v.pop_back();
76 }
77 string to_str() const {
78     string res;
79     if (v.empty()) return "0";
80     if (sign) res += '-';
81     res += to_string(v.back());
82     for (int i = (ll)v.size() - 2; i >= 0; i--) {
83         string add;
84         int w = v[i];
85         For(_,1,D){
86             add += ('0' + (w % 10));
87             w /= 10;
88         }
89         reverse(all(add));
90         res += add;
91     }
92     return res;
93 }
94 friend istream& operator>>(istream &is, BigInt &x) {
95     string tmp;
96     is >> tmp;
97     x = BigInt(tmp);
98     return is;
99 }
100 friend ostream& operator<<(ostream &os, BigInt x) {
101     os << x.to_str();
102     return os;
103 }
104
105
106 // ===== O(n) 运算部分 =====
107 // 高精度加法, 高精度减法, 高精度乘低精度, 高精度除低精度
108
109 BigInt& operator+=(const BigInt &x) {
110     if (sign != x.sign) {

```

```

111         *this -= (-x);
112         return *this;
113     }
114     if ((int)v.size() < (int)x.size())
115         v.resize(x.size(), 0);
116     For(i,0,(ll)x.size()-1)
117         v[i] += x.v[i];
118     norm();
119     return *this;
120 }
121 BigInt& operator--(const BigInt &x) {
122     if (sign != x.sign) {
123         *this += (-x);
124         return *this;
125     }
126     if (abs() < x.abs()) {
127         *this = x - (*this);
128         sign ^= 1;
129         return *this;
130     }
131     For(i,0,(ll)x.size()-1)
132         v[i] -= x.v[i];
133     norm();
134     return *this;
135 }
136 BigInt operator*(ll x) const { // 注意爆ll (D<=9)
137     BigInt res(*this);
138     if (x < 0) res.sign ^= 1, x *= -1;
139     for (int i = (ll)res.v.size()-1; i >= 0; i--)
140         res.v[i] *= x;
141     res.norm();
142     return res;
143 }
144 BigInt& operator/=(ll x) {
145     if (x < 0) sign ^= 1, x *= -1;
146     for (int i = (ll)v.size()-1; i >= 0; i--){
147         if (v[i] % x != 0 && i != 0) {
148             v[i - 1] += B * (v[i] % x);
149         }
150         v[i] /= x;
151     }
152     norm();
153     return *this;
154 }
155
156 BigInt operator+(const BigInt &x) const { return BigInt(*this) += x; }
157 BigInt operator-(const BigInt &x) const { return BigInt(*this) -= x; }
158 BigInt operator*(const int &x) { return (*this) = (*this)*x; }
159 BigInt operator/(const int &x) const { return BigInt(*this) /= x; }
160
161
162 // ===== 比较函数部分 =====
163
164 bool gtZer(){return sign==0 && v.size();} // return BitInt(x) > 0
165
166 bool operator<(const BigInt &x) const {

```

```

167         if (sign != x.sign) return sign > x.sign;
168         if (v.size() != x.size()) {
169             if (sign) return (int)x.size() < (int)v.size();
170             else return (int)v.size() < (int)x.size();
171         }
172         Rep(i, (ll)v.size()-1, 0) if (v[i] != x.v[i]) {
173             if (sign) return x.v[i] < v[i];
174             else return v[i] < x.v[i];
175         }
176         return false;
177     }
178     bool operator>(const BigInt &x) const { return x < *this; }
179     bool operator<=(const BigInt &x) const { return !(x < *this); }
180     bool operator>=(const BigInt &x) const { return !(*this < x); }
181     bool operator==(const BigInt &x) const { return !(*this < x) && !(x <
*this); }
182     bool operator!=(const BigInt &x) const { return !(*this == x); }
183
184
185     // ===== o(n^2) 运算部分 =====
186     // 高精度乘高精度, 高精度除高精度, 高精度模高精度, 高精度取模, 高精度平方
187
188     BigInt operator * (const BigInt &x) const {
189         #ifndef UseFFT
190             return n2Mul(*this, x);
191         #else
192             return FFTMul(*this, x);
193         #endif
194         // return karatsubaMul(x);
195     }
196     BigInt& operator/=(BigInt x){
197         int lstSign = sign ^ x.sign;
198         sign = x.sign = 0;
199         if ((*this) < x)
200             return *this = BigInt();
201         if (x == BigInt(1)){
202             sign = lstSign;
203             return *this;
204         }
205         int d = v.size() - x.size() + 1;
206         BigInt inv(1LL * B * B / x.v.back()), pre(0), c;
207         int cur = 2, bcur = 1;
208         while (inv != pre || bcur < x.size()) {
209             bcur = min(bcur << 1, x.size());
210             c.v = vector<ll>(x.v.end()-bcur, x.v.end());
211             pre = inv;
212             inv *= ((BigInt(2) << (cur + bcur - 1)) - inv * c);
213             cur = min(cur << 1, d);
214             inv.v = vector<ll>(inv.v.end()-cur, inv.v.end());
215         }
216         inv.v = vector<ll>(inv.v.end() - d, inv.v.end());
217         BigInt res = (*this) * inv;
218         res >>= (v.size());
219         BigInt tt = (*this) - res * x;
220         while (x <= tt) {
221             res += BigInt(1);

```



```

222         tt -= x;
223     }
224     v = res.v;
225     return *this;
226 }
227 BigInt& operator%=(const BigInt &x) {
228     BigInt div = (*this) / x;
229     (*this) -= div * x;
230     return *this;
231 }
232
233 void divMod(const BigInt& x, BigInt& divRes, BigInt& modRes) const {
234     divRes = (*this) / x;
235     modRes = (*this) - divRes*x;
236 }
237
238 BigInt square() {
239     BigInt res = *this;
240     res.sign = 0;
241     #ifdef UseFFT
242     auto v1 = FFT::vectorMul(v, v);
243     #else
244     auto v1 = n2Mul(*this, *this);
245     #endif
246     res.v.assign(v1.size(), 0);
247     For(i, 0, (ll)v1.size()-1) {
248         ll val = v1[i];
249         for (int j = i; val; j++) {
250             if (j == (int)res.v.size())
251                 res.v.push_back(0);
252             res.v[j] += val % B;
253             val /= B;
254         }
255     }
256     res.norm();
257     return res;
258 }
259
260 BigInt operator*=(const BigInt &x) { return (*this) = (*this)*x; }
261 BigInt operator/(const BigInt &x) const { return BigInt(*this) /= x; }
262 BigInt operator%(const BigInt &x) const { return BigInt(*this) %= x; }
263
264 private:
265
266     /*****三种乘法*****/
267     static BigInt n2Mul(const BigInt& a, const BigInt& b) {
268         BigInt res;
269         res.v.resize(a.size()+b.size(), 0);
270         res.sign = a.sign ^ b.sign;
271         for(int i = 0; i < a.size(); i++)
272             for(int j = 0; j < b.size(); j++)
273                 res[i+j] += a.v[i]*b.v[j];
274         res.norm();
275         return res;
276     }
277     // BigInt& karatsubaMul(const BigInt& x){

```

```

278 //      // todo
279 // }
280 #ifndef UseFFT
281 static BigInt FFTMul(const BigInt& a, const BigInt& b) {
282     if(a.size()+b.size() < 1000) return n2Mul(a, b);
283
284     BigInt res;
285     res.sign = a.sign ^ b.sign;
286     auto v1 = FFT::vectorMul(a.v, b.v);
287     res.v.assign(v1.size(), 0);
288     for(int i = 0; i < v1.size(); i++){
289         ll val = v1[i];
290         for (int j = i; val; j++) {
291             if (j == (int)res.v.size())
292                 res.v.push_back(0);
293             res.v[j] += val % B;
294             val /= B;
295         }
296     }
297     res.norm();
298     return res;
299 }
300 #endif
301
302 /** 这的左右移不是正常的左右移 做除法时用 */
303 BigInt& operator<=(const int& x){
304     if(!v.empty()) {
305         vector<ll> add(x, 0);
306         v.insert(v.begin(), all(add));
307     }
308     return *this;
309 }
310 BigInt& operator>=(const int& x){
311     v = vector<ll>(v.begin()+min(x,(int)v.size()), v.end());
312     return *this;
313 }
314 BigInt operator<<(const int& x) const {return BigInt(*this)<=x;}
315 BigInt operator>>(const int& x) const {return BigInt(*this)>=x;}
316 };
317 typedef BigInt Bigint;
318 Bigint qmi(Bigint m, int k)
319 {
320     Bigint res(1);
321     while (k) {
322         if (k & 1) res *= m;
323         m *= m;
324         k >>= 1;
325     }
326     return res;
327 }
328 }
329 using BigIntSP::BigInt;
330 using BigIntSP::qmi;

```

点分治

```
1  int n, k;
2  vector<pii> G[N];
3  // rt是重心
4  int vis[N];
5  int rt, sz[N], wt[N]; // wt[u]是u作为rt时的重儿子的sz
6  void findRt(int u, int fa, int subSZ){ // subSZ是当前子树大小
7      sz[u] = 1, wt[u] = 0;
8      for(auto [v, w]:G[u])if(v != fa && !vis[v]){
9          findRt(v, u, subSZ);
10         sz[u] += sz[v];
11         wt[u] = max(wt[u], sz[v]);
12     }
13     wt[u] = max(wt[u], subSZ - sz[u]);
14     if(wt[rt] > wt[u])rt = u; // 找个重儿子最小的作为根
15 }
16 int arr[N], cnt;
17 void dfs(int u, int fa, int dep){
18     arr[++cnt] = dep;
19     for(auto [v, w]:G[u])if(!vis[v] && v != fa)
20         dfs(v, u, dep+w);
21 }
22 // 算答案部分，点分治之后，算u子树内经过u的路径的答案
23 int calc(int u, int val){ // 计算子树内 路径+val <= k的对数，路径可能是非法的
24     cnt = 0;
25     dfs(u, 0, val);
26     sort(arr+1, arr+1+cnt);
27     int res = 0;
28     for(int l = 1, r = cnt; l <= r; l++){
29         while(l<=r && arr[l]+arr[r]>k)r--;
30         if(l>r)break;
31         // l -> u -> (l,r) 有r-l条
32         res += r-l;
33     }
34     return res;
35 }
36 int ans;
37 void DFS(int u){
38     ans += calc(u, 0);
39     vis[u] = 1; // vis设为1，做子树的时候就不会dfs出来了
40     for(auto [v, w]:G[u])if(!vis[v]){
41         ans -= calc(v, w); // 容斥 减掉前面calc(u, 0)多算的部分
42         rt = 0;
43         findRt(v, 0, sz[v]); // 重新在v子树找rt
44         DFS(rt);
45     }
46 }
47 void solve()
48 {
49     cin >> n;
50     ans = rt = 0;
51     wt[rt] = LINF;
52     For(i,1,n-1){
```

```
53     int u, v, w;
54     cin >> u >> v >> w;
55     G[u].push_back({v, w});
56     G[v].push_back({u, w});
57 }
58 cin >> k;
59 findRt(1, 0, n);
60 DFS(rt);
61 cout << ans << "\n";
62 }
```

树分治

树分治版题 (P6329)

给一棵树，每个点有点权，多次操作(强制在线)

1. 修改一个点的点权
2. 查询距离 x 小于等于 k 的所有点权和

复杂度 $O(n \log^2 n)$

```
1 // G1原树 G重构树
2 vector<int> G1[N], G[N];
3 BIT<ll> T[2][N]; // BIT用vector动态开时 查询修改记得取min
4 int fa[N][21], dep[N];
5 int cfa[N]; // 重构树上的fa
6 int n, a[N], q;
7 int rt, sz[N], vis[N], wt[N]; // wt[u]是u作为rt时的重儿子的sz
8 namespace LCASP{
9 void dfs0(int u, int f){
10     fa[u][0] = f;
11     For(i,1,20)fa[u][i] = fa[fa[u][i-1]][i-1];
12     for(auto v:G1[u])if(v != f){
13         dep[v] = dep[u]+1;
14         dfs0(v, u);
15     }
16 }
17 int lca(int u, int v){
18     if(dep[u] < dep[v])swap(u, v);
19     int cha = dep[u] - dep[v];
20     Rep(i,20,0)if(cha>>i&1)
21         u = fa[u][i];
22     if(u==v)return u;
23     Rep(i,20,0) if(fa[u][i] != fa[v][i]){
24         u = fa[u][i], v = fa[v][i];
25     }
26     return fa[u][0];
27 }
28 int dis(int u, int v){
29     int lc = lca(u, v);
30     return dep[u]+dep[v]-2*dep[lc];
31 }
32 }
33 using namespace LCASP;
34 void findRt(int u, int fa, int subSZ){ // subSZ是当前子树大小
35     sz[u] = 1, wt[u] = 0;
36     for(auto v:G1[u])if(v != fa && !vis[v]){
37         findRt(v, u, subSZ);
38         sz[u] += sz[v];
39         wt[u] = max(wt[u], sz[v]);
40     }
41     wt[u] = max(wt[u], subSZ - sz[u]);
42     if(wt[rt] > wt[u])rt = u; // 找个重儿子最小的作为根
43 }
```

```

44 int tsz[N];
45 void getsz(int u, int fa){
46     tsz[u] = 1;
47     for(auto v:G1[u])if(v != fa && !vis[v]){
48         getsz(v, u);
49         tsz[u] += tsz[v];
50     }
51 }
52 void dfs(int u) {
53     getsz(u, 0);    // 用findRt的sz是不准的(只是在点分治时负责度对)
54     T[0][u].tre.resize(tsz[u]+2, 0);
55     T[1][u].tre.resize(tsz[u]+2, 0);
56     vis[u] = 1;
57     for(auto v:G1[u])if(!vis[v]) {
58         rt = 0;
59         findRt(v, u, tsz[v]);
60         cfa[rt] = u;
61         G[u].push_back(rt); // 建重构树, 这题没用到
62         dfs(rt);
63     }
64 }
65 void solve()
66 {
67     cin >> n >> q;
68     For(i,1,n)cin >> a[i];
69     For(i,2,n) {
70         int u, v;
71         cin >> u >> v;
72         G1[u].push_back(v);
73         G1[v].push_back(u);
74     }
75     dfs0(1, 0);
76
77     rt = 0;
78     wt[0] = LINF;
79     findRt(1, 0, n);
80     dfs(rt);
81
82     auto add = [&](int u, int val){
83         for(int c = u; c; c=cfa[c]){
84             T[0][c].add(dis(u, c)+1, val);
85         }
86         for(int c = u; cfa[c]; c=cfa[c])T[1][c].add(dis(u, cfa[c])+1, val);
87     };
88     For(i,1,n) add(i, a[i]);
89
90     int ans = 0;
91     while(q--){
92         int op, x, y;
93         cin >> op >> x >> y;
94         x ^= ans, y ^= ans;
95         if(op == 0){
96             ans = T[0][x].query(y+1);
97             for(int u = x; cfa[u]; u = cfa[u]){
98                 int xdis = dis(cfa[u], x);
99                 if(y-xdis < 0)continue; // 这不能break 可能上面的点离x更近

```

```
100         ans += T[0][cfa[u]].query(y-xdis+1);
101         ans -= T[1][u].query(y-xdis+1);
102     }
103     cout << ans << "\n";
104 } else {
105     add(x, y-a[x]);
106     a[x] = y;
107 }
108 }
109 }
```

网络流封装

Dinic 最大流

```
1  template<typename T> struct Flow_ {
2      const int n;
3      const T inf = numeric_limits<T>::max();
4      struct Edge {
5          int to;
6          T w;
7          Edge(int to, T w) : to(to), w(w) {}
8      };
9      vector<Edge> ver;
10     vector<vector<int>> h;
11     vector<int> cur, d;
12
13     Flow_(int n) : n(n), h(n + 1) {}
14     void add(int u, int v, T c)
15     {
16         h[u].push_back(ver.size());
17         ver.emplace_back(v, c);
18         h[v].push_back(ver.size());
19         ver.emplace_back(u, 0);
20     }
21     bool bfs(int s, int t)
22     {
23         d.assign(n + 1, -1);
24         d[s] = 0;
25         queue<int> q;
26         q.push(s);
27         while(!q.empty())
28         {
29             auto x = q.front();
30             q.pop();
31             for(auto it: h[x])
32             {
33                 auto [y, w] = ver[it];
34                 if(w && d[y] == -1)
35                 {
36                     d[y] = d[x] + 1;
37                     if(y == t) return true;
38                     q.push(y);
39                 }
40             }
41         }
42         return false;
43     }
44     T dfs(int u, int t, T f)
45     {
46         if(u == t) return f;
47         auto r = f;
48         for(int &i = cur[u]; i < h[u].size(); i++)
49         {
50             auto j = h[u][i];
```



```

51         auto &[v, c] = ver[j];
52         auto &[u, rc] = ver[j ^ 1];
53         if(c && d[v] == d[u] + 1)
54         {
55             auto a = dfs(v, t, std::min(r, c));
56             c -= a;
57             rc += a;
58             r -= a;
59             if(!r) return f;
60         }
61     }
62     return f - r;
63 }
64 T work(int s, int t)
65 {
66     T ans = 0;
67     while(bfs(s, t))
68     {
69         cur.assign(n + 1, 0);
70         ans += dfs(s, t, inf);
71     }
72     return ans;
73 }
74 };
75 using Flow = Flow_<int>;

```

ISAP 最大流

```
1  template<typename T> struct Flow_ {
2      const int n;
3      const T inf = numeric_limits<T>::max();
4      struct Edge {
5          int to;
6          T cap, flow;
7          Edge(int _to = 0, T _cap = 0) :
8              to(_to), cap(_cap), flow(0) {}
9      };
10     vector<Edge> ver;
11     vector<vector<int>> h;
12     vector<int> dep, gap, cur;
13     vector<int> stk, que;
14
15     Flow_(int n) : n(n), h(n + 1) {}
16     void addedge(int u, int v, int w)
17     {
18         h[u].push_back(ver.size());
19         ver.emplace_back(v, w);
20         h[v].push_back(ver.size());
21         ver.emplace_back(u, 0);
22     }
23     void bfs(int s, int t)
24     {
25         dep.assign(n + 1, -1);
26         gap.assign(n + 2, 0);
27         que.assign(1, 0);
28         gap[0] = 1;
29         int hh = 0, tt = 0;
30         dep[t] = 0, que[0] = t;
31         while(hh <= tt)
32         {
33             int u = que[hh ++];
34             for(int i = 0; i < h[u].size(); i++)
35             {
36                 int j = h[u][i];
37                 auto &[v, c, f] = ver[j];
38                 if(dep[v] != -1) continue;
39                 if((int)que.size() == tt + 1) que.push_back(0);
40                 que[++ tt] = v;
41                 dep[v] = dep[u] + 1;
42                 gap[dep[v]] ++;
43             }
44         }
45     }
46     T work(int s, int t)
47     {
48         bfs(s, t);
49         cur.assign(n + 1, 0);
50         int u = s, top = 0;
51         T ans = 0;
52         while(dep[s] < n)
```

```

53     {
54         if(u == t)
55         {
56             T Min = inf;
57             int inser;
58             for(int i = 0; i < top; i++)
59                 if(Min > ver[stk[i]].cap - ver[stk[i]].flow)
60                 {
61                     Min = ver[stk[i]].cap - ver[stk[i]].flow;
62                     inser = i;
63                 }
64             for(int i = 0; i < top; i++)
65             {
66                 ver[stk[i]].flow += Min;
67                 ver[stk[i] ^ 1].flow -= Min;
68             }
69             ans += Min;
70             top = inser;
71             u = ver[stk[top] ^ 1].to;
72             continue;
73         }
74         bool flag = false;
75         int _v;
76         for(int &i = cur[u]; i < h[u].size(); i++)
77         {
78             auto j = h[u][i];
79             auto &[v, c, f] = ver[j];
80             if(c - f && dep[v] + 1 == dep[u])
81             {
82                 flag = true;
83                 _v = v;
84                 break;
85             }
86         }
87         if(flag)
88         {
89             if(stk.size() == top) stk.push_back(0);
90             stk[top++] = h[u][cur[u]];
91             u = _v;
92             continue;
93         }
94         int Min = n;
95         for(int i = 0; i < h[u].size(); i++)
96         {
97             auto j = h[u][i];
98             auto &[v, c, f] = ver[j];
99             if(c - f && dep[v] < Min)
100             {
101                 Min = dep[v];
102                 cur[u] = i;
103             }
104         }
105         gap[dep[u]]--;
106         if(!gap[dep[u]]) return ans;
107         dep[u] = Min + 1;
108         gap[dep[u]]++;

```

```
109         if(u != s) u = ver[stk[-- top] ^ 1].to;
110     }
111     return ans;
112 }
113 };
114 using Flow = Flow_<int>;
```

HLPP 预流推进

```
1  template<typename T> struct PushRelabel {
2      const int inf = 0x3f3f3f3f;
3      const T INF = 0x3f3f3f3f3f3f3f3f;
4      struct Edge {
5          int to, cap, flow, anti;
6          Edge(int v = 0, int w = 0, int id = 0) :
7              to(v), cap(w), flow(0), anti(id) {}
8      };
9      vector<vector<Edge>> e;
10     vector<vector<int>> gap;
11     vector<T> ex;
12     vector<bool> ingap;
13     vector<int> h;
14     int n, gobalcnt, maxH = 0;
15     T maxflow = 0;
16
17     PushRelabel(int n) : n(n), e(n + 1), ex(n + 1), gap(n + 1) {}
18     void addedge(int u, int v, int w)
19     {
20         e[u].push_back({v, w, (int)e[v].size()});
21         e[v].push_back({u, 0, (int)e[u].size() - 1});
22     }
23     void PushEdge(int u, Edge &edge)
24     {
25         int v = edge.to, d = min(ex[u], 1ll * edge.cap - edge.flow);
26         ex[u] -= d;
27         ex[v] += d;
28         edge.flow += d;
29         e[v][edge.anti].flow -= d;
30         if(h[v] != inf && d > 0 && ex[v] == d && !ingap[v])
31         {
32             ++ gobalcnt;
33             gap[h[v]].push_back(v);
34             ingap[v] = 1;
35         }
36     }
37     void PushPoint(int u)
38     {
39         for(auto k = e[u].begin(); k != e[u].end(); k++)
40         {
41             if(h[k->to] + 1 == h[u] && k->cap > k->flow)
42             {
43                 PushEdge(u, *k);
44                 if(!ex[u]) break;
45             }
46         }
47         if(!ex[u]) return;
48         if(gap[h[u]].empty())
49         {
50             for(int i = h[u] + 1; i <= min(maxH, n); i++)
51             {
52                 for(auto v: gap[i])
```

```

53         ingap[v] = 0;
54         gap[i].clear();
55     }
56 }
57 h[u] = inf;
58 for(auto [to, cap, flow, anti]: e[u])
59 {
60     if(cap > flow)
61         h[u] = min(h[u], h[to] + 1);
62 }
63 if(h[u] >= n) return;
64 maxH = max(maxH, h[u]);
65 if(!ingap[u])
66 {
67     gap[h[u]].push_back(u);
68     ingap[u] = 1;
69 }
70 }
71 void init(int t, bool f = 1)
72 {
73     ingap.assign(n + 1, 0);
74     for(int i = 1; i <= maxH; i++)
75         gap[i].clear();
76     globalcnt = 0, maxH = 0;
77     queue<int> q;
78     h.assign(n + 1, inf);
79     h[t] = 0, q.push(t);
80     while(q.size())
81     {
82         int u = q.front();
83         q.pop(), maxH = h[u];
84         for(auto &[v, cap, flow, anti]: e[u])
85         {
86             if(h[v] == inf && e[v][anti].cap > e[v][anti].flow)
87             {
88                 h[v] = h[u] + 1;
89                 q.push(v);
90                 if(f)
91                 {
92                     gap[h[v]].push_back(v);
93                     ingap[v] = 1;
94                 }
95             }
96         }
97     }
98 }
99 T work(int s, int t)
100 {
101     init(t, 0);
102     if(h[s] == inf) return maxflow;
103     h[s] = n;
104     ex[s] = INF;
105     ex[t] = -INF;
106     for(auto k = e[s].begin(); k != e[s].end(); k++)
107         PushEdge(s, *k);
108     while(maxH > 0)

```

```
109     {
110         if(gap[maxH].empty())
111         {
112             maxH --;
113             continue;
114         }
115         int u = gap[maxH].back();
116         gap[maxH].pop_back();
117         ingap[u] = 0;
118         PushPoint(u);
119         if(gobalcnt >= 10 * n)
120             init(t);
121     }
122     ex[s] -= INF;
123     ex[t] += INF;
124     return maxflow = ex[t];
125 }
126 };
```

EK 费用流

```
1 struct MinCostFlow {
2     using LL = long long;
3     using PII = pair<LL,int>;
4     const LL INF = numeric_limits<LL>::max();
5     struct Edge {
6         int v, c, f;
7         Edge(int v, int c, int f) : v(v), c(c), f(f) {}
8     };
9     const int n;
10    vector<Edge> e;
11    vector<vector<int>> g;
12    vector<LL> h, dis;
13    vector<int> pre;
14
15    MinCostFlow(int n) : n(n), g(n + 1) {}
16    void add(int u, int v, int c, int f) // c 流量, f 费用
17    {
18        g[u].push_back(e.size());
19        e.emplace_back(v, c, f);
20        g[v].push_back(e.size());
21        e.emplace_back(u, 0, -f);
22    }
23    bool dijkstra(int s, int t)
24    {
25        dis.assign(n + 1, INF);
26        pre.assign(n + 1, -1);
27        priority_queue<PII, vector<PII>, greater<PII>> que;
28        dis[s] = 0;
29        que.emplace(0, s);
30        while(!que.empty())
31        {
32            auto [d, u] = que.top();
33            que.pop();
34            if(dis[u] < d) continue;
35            for(int i: g[u])
36            {
37                auto [v, c, f] = e[i];
38                if(c > 0 && dis[v] > d + h[u] - h[v] + f)
39                {
40                    dis[v] = d + h[u] - h[v] + f;
41                    pre[v] = i;
42                    que.emplace(dis[v], v);
43                }
44            }
45        }
46        return dis[t] != INF;
47    }
48    pair<int,LL> flow(int s, int t)
49    {
50        int flow = 0;
51        LL cost = 0;
52        h.assign(n + 1, 0);
```



```

53     while(dijkstra(s, t))
54     {
55         for(int i = 1; i <= n; i++)
56             h[i] += dis[i];
57         int aug = numeric_limits<int>::max();
58         for(int i = t; i != s; i = e[pre[i] ^ 1].v)
59             aug = min(aug, e[pre[i]].c);
60         for(int i = t; i != s; i = e[pre[i] ^ 1].v)
61         {
62             e[pre[i]].c -= aug;
63             e[pre[i] ^ 1].c += aug;
64         }
65         flow += aug;
66         cost += LL(aug) * h[t];
67     }
68     return {flow, cost};
69 }
70 };

```

Bitset Bfs

bitset 相关函数

```
1 bitset<N> foo;
```

- `foo.size()` 返回大小 (位数)
- `foo.count()` 返回 1 的个数
- `foo.any()` 返回是否有 1
- `foo.none()` 返回是否没有 1
- `foo.set()` 全部置为 1
- `foo.set(p)` 将第 $p + 1$ 位置为 1
- `foo.set(p, x)` 将第 $p + 1$ 位置为 x
- `foo.reset()` 全部置为 0
- `foo.reset(p)` 将第 $p + 1$ 位置为 0
- `foo.flip` 全部取反
- `foo.flip(p)` 将第 $p + 1$ 位取反
- `foo.to_ulong()` 返回它转换为 `unsigned long` 的结果, 如果超出范围则报错
- `foo.to_ullong()` 返回它转换为 `unsigned long long` 的结果, 如果超出范围则报错
- `foo.to_string()` 返回它转换为 `string` 的结果

AcWing 164. 可达性统计

题意: 给定一张 n 个点 m 条边的有向无环图 ($1 \leq n, m \leq 30000$), 分别统计从每个点出发能到达的点的数量

解法1: 拓扑排序 + bitset

`bitset<N> f[N]` 维护每个点能到达的点集, 拓扑排序后倒序枚举每个点 `f[u] |= f[v]`

```
1 int n, m, d[N];
2 bitset<N> f[N];
3 vector<int> edges[N];
4 vector<int> top_que; // 拓扑序
5
6 void topsort()
7 {
8     queue<int> q;
9     for(int i = 1; i <= n; i++)
10         if(!d[i]) q.push(i);
11
12     while(q.size())
13     {
14         int u = q.front();
15         q.pop();
```

```

16     top_que.push_back(u);
17     for(auto v: edges[u])
18         if(!(--d[v])) q.push(v);
19 }
20 }
21
22 int main()
23 {
24     cin >> n >> m;
25     for(int i = 0; i < m; i++)
26     {
27         int u, v;
28         cin >> u >> v;
29         edges[u].push_back(v);
30         d[v] ++;
31     }
32
33     topsort();
34     for(int i = n - 1; i >= 0; i--)
35     {
36         int u = top_que[i];
37         f[u][u] = 1;
38         for(auto v: edges[u])
39             f[u] |= f[v];
40     }
41     for(int i = 1; i <= n; i++)
42         cout << f[i].count() << '\n';
43 }

```

解法2: 记忆化搜索

```

1  int n, m;
2  bitset<N> f[N];
3  vector<int> edges[N];
4  bool vis[N];
5
6  bitset<N> dp(int u)
7  {
8      if(vis[u]) return f[u];
9      vis[u] = true;
10
11      f[u][u] = 1;
12      for(auto v: edges[u])
13          f[u] |= dp(v);
14      return f[u];
15  }
16
17  int main()
18  {
19      cin >> n >> m;
20      for(int i = 0; i < m; i++)
21      {
22          int u, v;
23          cin >> u >> v;
24          edges[u].push_back(v);

```

```

25     }
26
27     for(int i = 1; i <= n; i++)
28         cout << dp(i).count() << '\n';
29 }

```

牛客练习赛14. 无向图中的最短距离

题意：有一个 n 个点, m 条边的无向图, 每条边的边权为 1, q 个询问, 每次询问给出若干个 (x_i, y_i) , 求至少与一个点对存在 $dist(x_i, v) \leq y_i$ 的点 v 的个数 ($n \leq 1000, m \leq 100000, q \leq 100000$)

题解: bitset + bfs + 可达性

bitset<N> dp[i][j] 维护 i 为起点, 距离 $\leq j$ 能到达的点集

cur 表示前 $j - 1$ 步的可达点集, nxt 表示前 j 步的可达点集, tmp 表示仅第 j 步可达的点集

tmp 可以由 $tem = cur \wedge nxt$ 得到, 当 $cur == nxt$ 时, 表示第 j 步无法再扩展, 此时 break 即可

lowBit 实现返回一段 bitset 内的 lowbit

```

1  const int maxn = 1010, maxd = 1 << 16 | 1;
2  int lbt[maxn];
3  void init(){
4      //lbt[i]表示i的二进制表示中lowbit在右起第几位(0-index) 用于lowbit函数的初始化
5      lbt[0] = -1;
6      for(int i = 1; i < maxd; ++i)
7          lbt[i] = i & 1 ? 0 : lbt[i >> 1] + 1;
8  }
9  //lowbit(bitset, low, upp) 求该bitset的[low, upp]内的lowbit
10 int lowBit(bitset<maxn> const &msk, size_t const &low, size_t const &upp) {
11     typedef unsigned long long _wordT;
12     _wordT *seq = (_wordT *)&msk;
13     size_t pL = low >> 6, pR = upp >> 6;
14     size_t qL = low & 63, qR = upp & 63;
15     for(size_t i = pL; i <= pR; ++i) {
16         _wordT val = seq[i];
17         if(i == pR && qR < 63)
18             val &= (static_cast<_wordT>(1) << (qR + 1)) - 1;
19         if(i == pL)
20             val = (val >> qL) << qL;
21         if(val != static_cast<_wordT>(0)) {
22             size_t ret = i << 6;
23             if((val & ((static_cast<_wordT>(1) << 32) - 1)) ==
static_cast<_wordT>(0)) {
24                 val >>= 32;
25                 ret |= 32;
26             }
27             if((val & ((static_cast<_wordT>(1) << 16) - 1)) ==
static_cast<_wordT>(0)) {
28                 val >>= 16;
29                 ret |= 16;
30             }

```

```

31         return ret + lbt[static_cast<int>(val & ((static_cast<_wordT>(1)
<< 16) - 1))]);
32     }
33 }
34 return -1;
35 }
36
37 int n, m, q;
38 bitset<maxn> e[maxn], dp[maxn][maxn], cur, nxt, tmp;
39
40 void solve()
41 {
42     cin >> n >> m >> q;
43     while(m --)
44     {
45         int u, v;
46         cin >> u >> v;
47         u --, v --;
48         e[u].set(v);
49         e[v].set(u);
50     }
51
52     init();
53     for(int i = 0; i < n; i++)
54     {
55         dp[i][0].set(i);
56         cur = dp[i][0];
57         nxt = cur | e[i];
58         for(int j = 1; j <= n; j++)
59         {
60             tmp = nxt ^ cur;    // 仅第 j 步可达
61             cur = dp[i][j] = nxt;    // 前 j 步可达
62
63             for(int u = lowBit(tmp, 0, n - 1); u != -1; u = lowBit(tmp, u +
1, n - 1))
64                 nxt |= e[u];    // 由仅第 j 步可达的点进行扩展
65             if(cur == nxt)
66             {
67                 for(int k = j + 1; k <= n; k++)
68                     dp[i][k] = cur;
69                 break;
70             }
71         }
72     }
73
74     while(q --)
75     {
76         int c, u, v;
77         cin >> c;
78         bitset<maxn> res;
79         while(c --)
80         {
81             cin >> u >> v;
82             u --;
83             res |= dp[u][v];
84         }

```

```
85         cout << res.count() << '\n';  
86     }  
87 }
```

封装 multiset 对顶堆

```
1 struct Set{
2     const int inf = 0x3f3f3f3f;
3     multiset<int> small, big;
4     Set() {
5         small.clear(), big.clear();
6         small.insert(-inf), big.insert(inf);
7     }
8     void adjust() {
9         while(small.size() > big.size() + 1)
10            {
11                multiset<int>::iterator it = (--small.end());
12                big.insert(*it);
13                small.erase(it);
14            }
15        while(big.size() > small.size())
16            {
17                multiset<int>::iterator it = big.begin();
18                small.insert(*it);
19                big.erase(it);
20            }
21    }
22    void add(int val) {
23        if(val <= *big.begin()) small.insert(val);
24        else big.insert(val);
25        adjust();
26    }
27    void del(int val) {
28        multiset<int>::iterator it = small.lower_bound(val);
29        if(it != small.end()) small.erase(it);
30        else
31            {
32                it = big.lower_bound(val);
33                big.erase(it);
34            }
35        adjust();
36    }
37    int get_middle() {
38        return *small.rbegin();
39    }
40 };
```

Miller-Rabin 大素数判定

Miller-Rabin 素性测试

- 作用: 快速判定一个数是否是质数
- 核心idea:

定理: 设 p 是奇质数, 存在正整数 s 与正奇数 d 使得 $p = 2^s d + 1$. 对于任意 $1 \leq a < p$, 满足要么 $a^d \equiv 1 \pmod{p}$, 要么存在一个 $0 \leq r < s$ 使得 $a^{2^r d} \equiv -1 \pmod{p}$

- 对于检验一个 $n \geq 3$ 的奇数是否为质数, 考虑一个随机算法: 随机一个 $1 \leq a < n$, 假如 a 满足上述定理的结论, 则称 a 是 n 的一个证人. 可以证明对于合数 n , 最多有 $n/4$ 个 "说谎的" 证人, 因此假设检验 t 次都符合结论, 则 n 是合数的概率最多为 $(1/4)^t$
- 时间复杂度: $O(t \log n)$

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  using i128 = __int128;
4
5  inline i128 read()
6  {
7      i128 x = 0, f = 1;
8      char ch = getchar();
9      while(ch < '0' || ch > '9')
10     {
11         if(ch == '-') f = -1;
12         ch = getchar();
13     }
14     while(ch >= '0' && ch <= '9')
15     {
16         x = (x << 1) + (x << 3) + (ch - '0');
17         ch = getchar();
18     }
19     return x * f;
20 }
21
22 void print(i128 x)
23 {
24     if(x < 0)
25     {
26         putchar('-');
27         x = -x;
28     }
29     if(x > 9) print(x / 10);
30     putchar(x % 10 + '0');
31 }
32
33 i128 mul(i128 a, i128 b, i128 p)
34 {
35     a %= p, b %= p;
36     i128 ans = 0;
37     while(b)
38     {
```



```

39     if(b & 1) ans = (ans + a) % p;
40     a = (a + a) % p;
41     b >>= 1;
42 }
43 return ans;
44 }
45
46 i128 qmi(i128 a, i128 b, i128 p)
47 {
48     i128 ans = 1;
49     while(b)
50     {
51         if(b & 1) ans = mul(ans, a, p);
52         a = mul(a, a, p);
53         b >>= 1;
54     }
55     return ans;
56 }
57
58 bool Miller_Rabin(i128 n)
59 {
60     if(n < 3 || n % 2 == 0) return n == 2;
61     if(n % 3 == 0) return n == 3;
62     i128 u = n - 1, t = 0;
63     while(u % 2 == 0) u /= 2, t++;
64
65     // test_time 为测试次数, 建议设为不小于 8 的整数
66     int test_time = 10;
67     for(int i = 0; i < test_time; i++)
68     {
69         // 0, 1, n-1 可以直接通过测试, a 取值范围 [2, n-2]
70         i128 a = rand() % (n - 3) + 2, v = qmi(a, u, n);
71         if(v == 1) continue;
72         int s;
73         for(s = 0; s < t; s++)
74         {
75             if(v == n - 1) break;
76             v = mul(v, v, n);
77         }
78         if(s == t) return 0;
79     }
80     return 1;
81 }
82
83 signed main()
84 {
85     i128 n = read();
86     if(Miller_Rabin(n)) puts("Yes");
87     else puts("No");
88 }

```

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using i128 = __int128;
4
5  namespace Miller_Rabin {
6      inline i128 read()
7      {
8          i128 x = 0, f = 1;
9          char ch = getchar();
10         while(ch < '0' || ch > '9')
11         {
12             if(ch == '-') f = -1;
13             ch = getchar();
14         }
15         while(ch >= '0' && ch <= '9')
16         {
17             x = (x << 1) + (x << 3) + (ch - '0');
18             ch = getchar();
19         }
20         return x * f;
21     }
22
23     void print(i128 x)
24     {
25         if(x < 0)
26         {
27             putchar('-');
28             x = -x;
29         }
30         if(x > 9) print(x / 10);
31         putchar(x % 10 + '0');
32     }
33
34     i128 mul(i128 a, i128 b, i128 p)
35     {
36         a %= p, b %= p;
37         i128 ans = 0;
38         while(b)
39         {
40             if(b & 1) ans = (ans + a) % p;
41             a = (a + a) % p;
42             b >>= 1;
43         }
44         return ans;
45     }
46
47     i128 qmi(i128 a, i128 b, i128 p)
48     {
49         i128 ans = 1;
50         while(b)
51         {
52             if(b & 1) ans = mul(ans, a, p);
53             a = mul(a, a, p);
54             b >>= 1;

```

```

55     }
56     return ans;
57 }
58
59 // 2, 7, 61 适用 4e9
60 // 2, 3, 5, 7, 11, 13, 17 适用 3e14
61 // 2, 3, 7, 61, 24251 适合 1e16 特判 46856248255981
62 static constexpr int primelist[] = {2, 325, 9375, 28178, 450775,
9780504, 1795265022};
63 bool miller_rabin(i128 n)
64 {
65     if (n < 3 || n % 2 == 0) return n == 2;
66     i128 a = n - 1, b = 0;
67     while (a % 2 == 0) a >>= 1, b++;
68
69     for (int i : primelist)
70     {
71         i128 v = qmi(i, a, n);
72         if(v == 1) continue;
73         for (int j = 0; j < b || (b == -1, false); j++)
74         {
75             if(v == n - 1) break;
76             v = mul(v, v, n);
77         }
78         if (b < 0) return false;
79     }
80     return true;
81 }
82 }
83 using namespace Miller_Rabin;
84
85 signed main()
86 {
87     i128 n = read();
88     if(miller_rabin(n)) puts("Yes");
89     else puts("No");
90 }

```

Pollard rho 大整数分解

Pollard rho 算法

- 作用: 快速地分解一个整数
- 假设当前有一个合数 n
- 核心 idea:

idea: 给定一个模 n 意义下的多项式 $g(x)$, (比如选取 $g(x) = (x^2 + c) \bmod n$), 它充当一个伪随机数生成器. 选定任一 $0 \leq x_0 < n$, 用 $g(x)$ 生成一个序列 $x_i = g(x_{i-1}), (i > 0)$. 序列 x_i 会循环, 假设 p 是 n 的一个因子, 则 $\{x_i \bmod p\}$ 也会成环, 且环长只可能更短

- 复杂度: $O(n^{1/4})$

```
1 namespace Let_it_Rot {
2     typedef long long ll;
3     using f64 = long double;
4
5     ll p;
6     f64 invp;
7
8     void setmod(ll x)
9     {
10         p = x, invp = (f64) 1 / x;
11     }
12
13     ll mul(ll a, ll b)
14     {
15         ll z = a * invp * b + 0.5;
16         ll res = a * b - z * p;
17         return res + (res >> 63 & p);
18     }
19
20     ll pow(ll a, ll x, ll res = 1)
21     {
22         for(; x >>= 1, a = mul(a, a))
23             if(x & 1) res = mul(res, a);
24         return res;
25     }
26
27     bool checkprime(ll p)
28     {
29         if(p == 1) return 0;
30         setmod(p);
31         ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
32         for(ll a: {2, 3, 5, 7, 11, 13, 82, 373})
33         {
34             if(a % p == 0) continue;
35             ll x = pow(a, s), y;
36             for(int i = 0; i < d; i++, x = y)
37             {
38                 y = mul(x, x);
39                 if(y == 1 && x != 1 && x != p - 1)
```

```

40         return 0;
41     }
42     if(x != 1) return 0;
43 }
44 return 1;
45 }
46
47 ll rho(ll n)
48 {
49     if(!(n & 1)) return 2;
50     static std::mt19937_64 gen(((size_t)"evilboy"));
51     ll x = 0, y = 0, prod = 1;
52     auto f = [&](ll o) {return mul(o, o) + 1;};
53     setmod(n);
54     for(int t = 30, z = 0; t % 64 || std::__gcd(prod, n) == 1; t++)
55     {
56         if(x == y) x = ++z, y = f(x);
57         if(ll q = mul(prod, x + n - y)) prod = q;
58         x = f(x), y = f(f(y));
59     }
60     return std::__gcd(prod, n);
61 }
62 std::vector<ll> factor(ll x)
63 {
64     std::vector<ll> res;
65     auto f = [&](auto f, ll x)
66     {
67         if(x == 1) return;
68         if(checkprime(x)) return res.push_back(x);
69         ll y = rho(x);
70         f(f, y), f(f, x / y);
71     };
72     f(f, x), sort(res.begin(), res.end());
73     return res;
74 }
75 }
76 using namespace Let_it_Rot;

```

双模哈希封装

```
1 namespace Hash {
2     const int N = 1000010;
3     typedef unsigned long long ull;
4     typedef pair<ull,ull> puu;
5     ull base = 131;
6     ull mod1 = 25165843, mod2 = 1610612741;
7     ull bs1[N], bs2[N];
8
9     void init() {
10         bs1[0] = bs2[0] = 1;
11         for(int i = 1; i < N; i++) {
12             bs1[i] = bs1[i - 1] * base % mod1;
13             bs2[i] = bs2[i - 1] * base % mod2;
14         }
15     }
16
17     struct String
18     {
19         int n;
20         string s;
21         vector<ull> h1, h2;
22
23         String(string _s) {
24             n = _s.size();
25             s = " " + _s;
26             h1.resize(n + 1, 0), h2.resize(n + 1, 0);
27             for(int i = 1; i <= n; i++) {
28                 h1[i] = (h1[i - 1] * base % mod1 + (ull)s[i]) % mod1;
29                 h2[i] = (h2[i - 1] * base % mod2 + (ull)s[i]) % mod2;
30             }
31         }
32         ull get_hash1(int l, int r) {
33             return (h1[r] - h1[l - 1] * bs1[r - l + 1] % mod1 + mod1) %
mod1;
34         }
35         ull get_hash2(int l, int r) {
36             return (h2[r] - h2[l - 1] * bs2[r - l + 1] % mod2 + mod2) %
mod2;
37         }
38         puu get_hash(int l, int r) {
39             return {get_hash1(l, r), get_hash2(l, r)};
40         }
41         puu get_hash(void) {
42             return get_hash(1, n);
43         }
44     };
45 }
46 using namespace Hash;
```

哈希 + 数据结构 (卡自然溢出)

哈希类

```
1 namespace Hash {
2     typedef unsigned long long ull;
3     const int N = 500010;
4     ull base = 13131, bas[N];
5
6     inline void init_base() {
7         bas[0] = 1;
8         for(int i = 1; i < N; i++)
9             bas[i] = bas[i - 1] * base;
10    }
11
12    struct HashNode {
13        ull val;
14        int len;
15        HashNode() {}
16        HashNode(ull a, int b): val(a), len(b) {}
17    };
18    inline HashNode operator + (const HashNode& a, const HashNode &b) {
19        HashNode res;
20        res.val = a.val * bas[b.len] + b.val;
21        res.len = a.len + b.len;
22        return res;
23    }
24    inline HashNode operator - (const HashNode& a, const HashNode &b) {
25        HashNode res;
26        res.val = a.val - b.val * bas[a.len - b.len];
27        res.len = a.len - b.len;
28        return res;
29    }
30    inline bool operator == (const HashNode& a, const HashNode& b) {
31        return (a.len == b.len) && (a.val == b.val);
32    }
33 }
```

动态字符串哈希 (线段树维护字符串哈希)

$O(\log n)$ 单点修改, $O(\log n)$ 区间查询

```
1 namespace Hash {
2     typedef unsigned long long ull;
3     const int N = 500010;
4     ull base = 13131, bas[N];
5
6     inline void init_base() {
7         bas[0] = 1;
8         for(int i = 1; i < N; i++)
9             bas[i] = bas[i - 1] * base;
```

```

10     }
11
12     struct HashNode {
13         ull val;
14         int len;
15         HashNode(){}
16         HashNode(ull a, int b): val(a), len(b) {}
17     };
18     inline HashNode operator + (const HashNode& a, const HashNode &b) {
19         HashNode res;
20         res.val = a.val * bas[b.len] + b.val;
21         res.len = a.len + b.len;
22         return res;
23     }
24     inline HashNode operator - (const HashNode& a, const HashNode &b) {
25         HashNode res;
26         res.val = a.val - b.val * bas[a.len - b.len];
27         res.len = a.len - b.len;
28         return res;
29     }
30     inline bool operator == (const HashNode& a, const HashNode& b) {
31         return (a.len == b.len) && (a.val == b.val);
32     }
33
34     HashNode tr[N << 2];
35     inline void pushup(int u) {
36         tr[u] = tr[u << 1] + tr[u << 1 | 1];
37     }
38     void build(int u, int l, int r, const string& str) {
39         if(l == r) {
40             tr[u] = HashNode(str[l] - 'a' + 1, 1);
41             return;
42         }
43         int mid = l + r >> 1;
44         build(u << 1, l, mid, str), build(u << 1 | 1, mid + 1, r, str);
45         pushup(u);
46     }
47     void update(int u, int l, int r, int x, char ch) {
48         if(l == r) {
49             tr[u] = HashNode(ch - 'a' + 1, 1);
50             return;
51         }
52         int mid = l + r >> 1;
53         if(x <= mid) update(u << 1, l, mid, x, ch);
54         else update(u << 1 | 1, mid + 1, r, x, ch);
55         pushup(u);
56     }
57     HashNode query(int u, int l, int r, int L, int R) {
58         if(l >= L && r <= R) return tr[u];
59         int mid = l + r >> 1;
60         HashNode res(0, 0);
61         if(L <= mid) res = res + query(u << 1, l, mid, L, R);
62         if(R > mid) res = res + query(u << 1 | 1, mid + 1, r, L, R);
63         return res;
64     }
65 }

```


线段树维护正反串哈希

$O(\log n)$ 单点修改, $O(\log n)$ 区间查询. 用于判断区间是否为回文串

```
1 namespace Hash {
2     typedef unsigned long long ull;
3     const int N = 1000010;
4     ull base = 13131, bas[N];
5
6     inline void init_base() {
7         bas[0] = 1;
8         for(int i = 1; i < N; i++)
9             bas[i] = bas[i - 1] * base;
10    }
11
12    struct HashNode {
13        ull pval, sval; // 正串哈希, 反串哈希
14        int len;
15        HashNode(){}
16        HashNode(ull a, ull b, int c): pval(a), sval(b), len(c) {}
17    };
18    inline HashNode operator + (const HashNode& a, const HashNode &b) {
19        HashNode res;
20        res.pval = a.pval * bas[b.len] + b.pval;
21        res.sval = b.sval * bas[a.len] + a.sval;
22        res.len = a.len + b.len;
23        return res;
24    }
25
26    HashNode tr[N << 2];
27    inline void pushup(int u) {
28        tr[u] = tr[u << 1] + tr[u << 1 | 1];
29    }
30    void build(int u, int l, int r, const string& str) {
31        if(l == r) {
32            tr[u] = HashNode(str[l] - 'a' + 1, str[l] - 'a' + 1, 1);
33            return;
34        }
35        int mid = l + r >> 1;
36        build(u << 1, l, mid, str), build(u << 1 | 1, mid + 1, r, str);
37        pushup(u);
38    }
39    void update(int u, int l, int r, int x, char ch) {
40        if(l == r) {
41            tr[u] = HashNode(ch - 'a' + 1, ch - 'a' + 1, 1);
42            return;
43        }
44        int mid = l + r >> 1;
45        if(x <= mid) update(u << 1, l, mid, x, ch);
46        else update(u << 1 | 1, mid + 1, r, x, ch);
47        pushup(u);
48    }
```

```
49     HashNode query(int u, int l, int r, int L, int R) {
50         if(l >= L && r <= R) return tr[u];
51         int mid = l + r >> 1;
52         HashNode res(0, 0, 0);
53         if(L <= mid) res = res + query(u << 1, l, mid, L, R);
54         if(R > mid) res = res + query(u << 1 | 1, mid + 1, r, L, R);
55         return res;
56     }
57 }
```

树与图上的计数问题

Prufer Code

n 个点的有标号无根树可以与一个长度为 $n - 2$ 的 *Prufer* 序列对应

从树到 Prufer 序列

- f 为空序列
- 如果当前树上多于两个节点, 假设当前标号最小的叶子为 x , 与 x 相连的节点标号为 y , 把节点 x 从树上删掉, 把 y 放到序列 f 的末尾
- 重复上述操作, 直到树上只有两个节点

```
1 vector<int> tree_to_prufer(vector<set<int>> e)
2 {
3     int n = e.size();
4     vector<int> f;
5     priority_queue<int, vector<int>, greater<int>> leaves;
6     for(int i = 0; i < n; i++)
7         if(e[i].size() == 1)
8             leaves.push(i);
9
10    while(f.size() < n - 2)
11    {
12        int u = leaves.top();
13        leaves.pop();
14
15        int v = *e[u].begin();
16        e[v].erase(u);
17        f.push_back(v);
18        if(e[v].size() == 1) leaves.push(v);
19    }
20    return f;
21 }
```

从 Prufer 序列到树

- 找到当前不在 f 中且还没被使用过的最小的元素 x
- x 与当前 f 的第一个元素连边, 把 x 标记为已使用过
- 删除 f 的第一个元素, 如果 f 非空重复上述操作
- 最终有两个元素没被使用, 将它们连边

```
1 vector<vector<int>> prufer_to_tree(vector<int> f)
2 {
3     int n = f.size() + 2;
4     vector<vector<int>> edges(n, vector<int> ());
5     vector<int> cnt(n, 0);
6     for(int u: f) cnt[u] ++;
7     priority_queue<int, vector<int>, greater<int>> q;
8     for(int i = 0; i < n; i++) if(!cnt[i]) q.push(i);
9
10    for(int u: f)
```

```

11     {
12         int v = q.top();
13         q.pop();
14         edges[u].push_back(v);
15         edges[v].push_back(u);
16         cnt[u]--;
17         if(!cnt[u]) q.push(u);
18     }
19     int u = q.top(); q.pop();
20     int v = q.top(); q.pop();
21     edges[u].push_back(v);
22     edges[v].push_back(u);
23     return edges;
24 }

```

上述两个映射定义了 n 个点有标号无根树的集合到长度为 $n - 2$, 元素在 1 到 n 之间的序列的一个一一映射

定理 (Cayley 定理)

n 个点的有标号无根树有 n^{n-2} 种

matrix-tree theorem

定理

设无向图 $G(V, E)$, $D = \text{diag}(d(1), d(2), \dots, d(n))$, $d(i)$ 是节点 i 的度数, G 的拉普拉斯矩阵 $L = D - E$. G 的生成树个数为 $\det(L_0)$, 其中 L_0 是 L 去掉第 i 行第 i 列 (i 任选)

SA 封装

```
1 namespace SA {
2     const int N = 1000010;
3     int n, m; //字符串长度、不同字符数量
4     string s;
5     //sa[i] 表示排名第 i 的是第几个后缀
6     //x[i] 表示第 i 个后缀的第一关键字
7     //y[i] 表示第 i 个后缀的第二关键字
8     //c[i] 表示关键字为 i 的数的个数
9     //rk[i] 表示第 i 个后缀的排名
10    //height[i] 表示排名第 i 的后缀和排名第 i - 1 的后缀的最长公共前缀
11    int sa[N], x[N], y[N], c[N], rk[N], height[N];
12
13    void get_sa() //预处理 sa
14    {
15        //将所有后缀按照首字母从小到大排序（基数排序）
16        for(int i = 1; i <= n; i++) c[x[i] = s[i]]++; //记录每个关键字出现的次数
17        for(int i = 2; i <= m; i++) c[i] += c[i - 1]; //求前缀和
18        for(int i = n; i >= 1; i--) sa[c[x[i]]--] = i; //为每个数安排位置
19
20        //每一轮将后缀按照前 2k 个字符排序
21        for(int k = 1; k <= n; k <= 1)
22        {
23            //先将所有后缀按照第二关键字排序
24            int num = 0;
25            for(int i = n - k + 1; i <= n; i++) y[++num] = i;
26            for(int i = 1; i <= n; i++)
27                if(sa[i] > k)
28                    y[++num] = sa[i] - k;
29
30            //再将所有后缀按照第一关键字排序（基数排序）
31            for(int i = 1; i <= m; i++) c[i] = 0;
32            for(int i = 1; i <= n; i++) c[x[i]]++;
33            for(int i = 2; i <= m; i++) c[i] += c[i - 1];
34            for(int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i], y[i] = 0;
35            //y 数组清空，用于后面存储当前的第一关键字
36
37            //将排序后的所有后缀再按照前 2k 个字符离散化
38            swap(x, y);
39            x[sa[1]] = 1, num = 1;
40            for(int i = 2; i <= n; i++)
41                x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] ==
42                    y[sa[i - 1] + k]) ? num : ++num;
43
44            if(num == n) break; //如果 n 个后缀的排名都不同，说明已经排好序
45            m = num; //更新不同字符的个数
46        }
47    }
48
49    void get_height() //预处理 height
50    {
51        for(int i = 1; i <= n; i++) rk[sa[i]] = i; //预处理 rk
52        //预处理 height
```

```

51     for(int i = 1, k = 0; i <= n; i++) //k 记录当前的 h[i]
52     {
53         if(rk[i] == 1) continue; //height[1] 默认为 0
54         if(k) k--; //应该从 h[i - 1] - 1 开始枚举, 也就是 k - 1
55         int j = sa[rk[i] - 1]; //记录第 i 个后缀前一个排名的后缀
56         //如果 i 和 j 的第 k 位相同, 则最长公共前缀长度 + 1
57         while(i + k <= n && j + k <= n && s[i + k] == s[j + k]) k++;
58         height[rk[i]] = k; //height[rk[i]] = k
59     }
60 }
61
62 void init()
63 {
64     s = " " + s;
65     n = s.size() - 1, m = 122; //最大字符 'z' 的 ASCII 码是 122
66     get_sa(); //预处理 sa
67     get_height(); //预处理 height
68 }
69 };
70 using namespace SA;

```

排列组合公式

排列数公式

$$A_n^m = n(n-1)(n-2)\cdots(n-m+1) = \frac{n!}{(n-m)!}$$

排列数性质

$$A_n^m = nA_{n-1}^{m-1}$$

$$A_n^m = mA_{n-1}^{m-1} + A_{n-1}^m$$

组合数公式

$$C_n^m = \frac{n(n-1)(n-2)\cdots(n-m+1)}{m!} = \frac{n!}{m!(n-m)!}$$

$$C_n^0 = C_n^n = 1$$

组合数性质

$$C_n^m = C_n^{n-m}$$

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$$

组合数求和公式

$$C_n^0 + C_n^1 + C_n^2 + \cdots + C_n^n = 2^n$$

$$C_n^0 + C_n^2 + C_n^4 + \cdots = C_n^1 + C_n^3 + C_n^5 + \cdots = 2^{n-1}$$

二项式定理

$$(1+x)^n = \sum_{k=0}^n \binom{n}{k} x^k = \binom{n}{0} + \binom{n}{1}x + \cdots + \binom{n}{n}x^n$$

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k$$

二项式系数

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = 1, \binom{0}{k} = 0$$

有关二项式系数的恒等式

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

$$\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$$

$$\binom{n-1}{k} - \binom{n-1}{k-1} = \frac{n-2k}{n} \binom{n}{k}$$

$$\binom{n}{i} \binom{i}{m} = \binom{n}{m} \binom{n-m}{i-m}$$

$$\sum_{r=0}^n \binom{n}{r} = 2^n$$

$$\sum_{r=0}^{n-k} \frac{(-1)^r (n+1)}{k+r+1} \binom{n-k}{r} = \binom{n}{k}^{-1}$$

$$\sum_{r=0}^n \binom{dn}{dr} = \frac{1}{d} \sum_{r=1}^d (1 + e^{\frac{2\pi ri}{d}})^{dn}$$

$$\sum_{i=m}^n \binom{a+i}{i} = \binom{a+n+1}{n} - \binom{a+m}{m-1}$$

$$\binom{a+m}{m-1} + \binom{a+m}{m} + \binom{a+m+1}{m+1} + \cdots + \binom{a+n}{n} = \binom{a+n+1}{n}$$

$$F_n = \sum_{i=0}^{\infty} \binom{ni}{i}$$

$$F_{n-1} + F_n = \sum_{i=0}^{\infty} \binom{n-1-i}{i} + \sum_{i=0}^{\infty} \binom{n-i}{i} = 1 + \sum_{i=1}^{\infty} \binom{n-i}{i-1} + \sum_{i=1}^{\infty} \binom{n-i}{i} = 1$$

$$\sum_{i=0}^{\infty} \binom{n+1-i}{i} = F_{n+1}$$

朱世杰恒等式

$$\sum_{i=m}^n \binom{i}{a} = \binom{n+1}{a+1} - \binom{m}{a+1}$$

$$\binom{m}{a+1} + \binom{m}{a} + \binom{m+1}{a} + \cdots + \binom{n}{a} = \binom{n+1}{a+1}$$

二阶求和公式

$$\sum_{r=0}^n \binom{n}{r}^2 = \binom{2n}{n}$$

$$\sum_{i=0}^r \binom{r_1+n-1-i}{r_1-1} \binom{r_2+i-1}{r_2-1} = \binom{r_1+r_2+n-1}{r_1+r_2-1}$$

$$(1-x)^{-r_1} (1-x)^{-r_2} = (1-x)^{-r_1-r_2}$$

$$(1-x)^{-r_1} (1-x)^{-r_2} = \left(\sum_{n=0}^{\infty} \binom{r_1+n-1}{r_1-1} x^n\right) \left(\sum_{n=0}^{\infty} \binom{r_2+n-1}{r_2-1} x^n\right) = \sum_{n=0}^{\infty} \left(\sum_{i=0}^n \binom{r_1+n-1-i}{r_1-1} \binom{r_2+i-1}{r_2-1}\right) x^n$$

$$(1-x)^{-r_1-r_2} = \sum_{n=0}^{\infty} \binom{r_1+r_2+n-1}{r_1+r_2-1} x^n$$

范德蒙恒等式

$$\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}$$

三阶求和公式 李善兰恒等式

$$\binom{n+k}{k}^2 = \sum_{j=0}^k \binom{k}{j}^2 \binom{n+2k-j}{2k}$$



北京大学

PEKING UNIVERSITY

Let it Rot

Coach

张勤健

罗国杰

Qinjian Zhang

Guojie Luo

Contestant

钱易

彭博

冯施源

Yi Qian

Bo Peng

Shiyuan Feng

ICPC World Finals Luxor

Contents

1	图论
1.1	欧拉回路
1.2	二分图匹配 最小边覆盖
1.3	网络最大流 dinic
1.4	最小费用流
1.5	二分图最大权匹配 KM
1.6	一般图最大匹配 带花树
1.7	最小树形图
1.8	缩点 kasaraju
1.9	缩点 Tarjan
1.10	缩点 点双
1.11	缩点 边双
1.12	仙人掌
1.13	2-Sat
1.14	支配树
1.15	三/四元环
1.16	双极定向
1.17	Tree And Graph
1.17.1	树的计数, Prufer 序列
1.17.2	有根树的计数
1.17.3	无根树的计数
1.17.4	生成树计数 Kirchhoff's Matrix-Tree Theorem
1.17.5	有向图欧拉回路计数 BEST Theorem
1.17.6	Tutte Matrix
1.17.7	Edmonds Matrix
1.18	拟阵交
2	数论
2.1	取模还原分数
2.2	扩展欧几里得
2.3	万能欧几里得
2.4	直线下点数欧几里得
2.5	Stem-Brocot Tree 二分
2.6	扩展中国剩余定理
2.7	Miller-Rabin
2.8	Pollard-rho
3	Math
3.1	拉格朗日反演
3.2	分拆数/五边形数
3.3	Fast Fourier Transform
3.4	Number Theoretic Transform
3.5	Generating function
3.6	全在线卷积
3.7	Bertekamp Massey
3.8	线性规划 单纯形法
3.9	Simpson 积分
3.10	黄金三分
4	字符串
4.1	后缀自动机 SAM
4.2	基本字符串字典
4.3	DAG 剖分
4.4	exKMP
4.5	log 个最小后缀
4.6	SA
4.7	PAM
4.8	AC 自动机
4.9	Manacher
4.10	Lyndon/最小表示法

4.11	Runs	16
5	数据结构	16
5.1	区间/加区间求和和树状数组	16
5.2	zkw 线段树	16
5.3	Link Cut Tree	17
5.4	FHQ Treap	17
5.5	pbds tree	17
6	geometry	17
6.1	向量	17
6.2	直线/半平面	18
6.3	半平面交	18
6.4	线段	18
6.5	多边形	19
6.6	线段 in 多边形	19
6.7	图形求交	19
6.8	凸包	20
6.9	上凸壳	20
6.10	最小圆覆盖	20
6.11	最近点对	21
6.12	凸包直径	21
6.13	切凸包	21
6.14	V 图	21
6.15	DeLaunay 三角剖分	21
7	geometry3d	22
7.1	向量	22
7.2	平面	23
7.3	直线	23
7.4	凸包	23
8	Misc	23
8.1	Pragma	23
8.2	Barrett	23
8.3	ICS	23
8.4	日期公式	24
8.5	Xorshift	24
9	配置	24
9.1	vimrc	24
9.2	bashrc	24
9.3	对拍	24
9.4	编译参数	24
9.5	随机素数	24
9.6	常数表	24
10	注意事项	24
10.1	测试项目	24
10.2	bugs	24
11	tables	25
11.1	导数积分	25
1	图论	
1.1	欧拉回路	
<div>namespace Euler { bool directed; vector<pi>V[<i>sz</i>]; }</div>		

```

4 | vector<int>ans; // reverse ans in the end
5 | int vis[intsz];
6 | int dfs(int x) {
7 |     vector<int>t;
8 |     while (V[x].size()) {
9 |         auto [to, id]=V[x].back();
10 |         V[x].pop_back();
11 |         if (!vis[abs(id)])
12 |             ↪ vis[abs(id)]=1, t.push_back(dfs(to)), ans.push_back(id);
13 |     }
14 |     rep(i, 1, (int)t.size()-1) if (t[i]!=x) ans.clear();
15 |     return t.size()?t[0]:x;
16 | }
17 | int n, m;
18 | int e[intsz];
19 | void clr() {
20 |     rep(i, 1, n) V[i].clear(), deg[i]=vv[i]=0;
21 |     rep(i, 1, m) vis[i]=0;
22 |     ans.clear();
23 |     n=m=0;
24 | }
25 | void addedge(int x, int y) {
26 |     chkmax(n, x), chkmax(m, y); ++m;
27 |     e[m]={x, y};
28 |     if (directed) {
29 |         V[x].push_back({y, m});
30 |         ++deg[x], --deg[y], vv[x]=vv[y]=1;
31 |     }
32 |     else {
33 |         V[x].push_back({y, m});
34 |         V[y].push_back({x, -m});
35 |         ++deg[x], ++deg[y], vv[x]=vv[y]=1;
36 |     }
37 | }
38 | using vi=vector<int>;
39 | pair<vi, vi> work() {
40 |     if (!m) return clr(), pair<vi, vi>{{1}, {}};
41 |     int S=1;
42 |     rep(i, 1, n) if (vv[i]) S=i;
43 |     rep(i, 1, n) if (deg[i]>0&&deg[i]%2==1) S=i;
44 |     dfs(S);
45 |     if ((int)ans.size()!=m) return clr(), pair<vi, vi>();
46 |     reverse(ans.begin(), ans.end());
47 |     vi ver, edge=ans;
48 |     if (directed) {
49 |         ver={e[ans[0]].fir};
50 |         for (auto t:ans) ver.push_back(e[t].sec);
51 |     }
52 |     else {
53 |         ver={ans[0]>0?e[ans[0]].fir:e[ans[0]].sec};
54 |         for (auto t:ans) ver.push_back(t>0?t[t].sec:e[-t].fir);
55 |     }

```

```

56 |     clr();
57 |     return {ver, edge};
58 | }
59 | }

```

1.2 二分图匹配 | 最小边覆盖

// 匈牙利, 左到右单向边, $O(M|match|)$

```

1 | std::vector<int> edge[N];
2 | bool dfs(int x, std::vector<int> & vis, std::vector<int> & match) {
3 |     for(int y : edge[x]) if(!vis[y])
4 |         if(vis[y] = 1, !match[y] || dfs(match[y], vis, match))
5 |             | return match[y] = x, 1;
6 |     return 0;
7 | }
8 | std::vector<int> match(int nl, int nr) {
9 |     std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
10 |    for(int i = 1; i <= nl; ++i) if(dfs(i, vis, match))
11 |        | memset(vis.data(), 0, vis.size() << 2);
12 |    for(int i = 1; i <= nr; ++i) ret[match[i]] = i;
13 |    return ret[0] = 0, ret;
14 | }
15 |
16 | // 最小边覆盖
17 | std::pair<std::vector<int>, std::vector<int>> minedgecover(int nl, int nr) {
18 |     std::vector<int> vis(nr + 1), match(nr + 1), ret(nl + 1);
19 |     for(int i = 1; i <= nl; ++i) if(dfs(i, vis, match))
20 |         | memset(vis.data(), 0, vis.size() << 2);
21 |     for(int i = 1; i <= nr; ++i) ret[match[i]] = i;
22 |     ret[0] = 0;
23 |     for(int i = 1; i <= nl; ++i) if(!ret[i]) dfs(i, vis, match);
24 |     std::vector<int> le, ri;
25 |     for(int i = 1; i <= nl; ++i) if(ret[i] && !vis[ret[i]]) le.push_back(i);
26 |     for(int i = 1; i <= nr; ++i) if(vis[i]) ri.push_back(i);
27 |     return std::make_pair(le, ri);
28 | }

```

// 匈牙利, 左到右单向边, $bitset$, $O(n^2/w|match|)$

```

29 | using set = std::bitset<N>;
30 | set edge[N];
31 | bool dfs(int x, set & unvis, std::vector<int> & match) {
32 |     for(set z = edge[x];;) {
33 |         | z &= unvis;
34 |         | int y = z._find_first();
35 |         | if(y == N) return 0;
36 |         | if(unvis.reset(y), !match[y] || dfs(match[y], unvis, match))
37 |             | return match[y] = x, 1;
38 |         | }
39 |     }
40 | }
41 | std::vector<int> match(int nl, int nr) {
42 |     set unvis; unvis.set();
43 |     std::vector<int> match(nr + 1), ret(nl + 1);
44 |     for(int i = 1; i <= nl; ++i)
45 |         | if(dfs(i, unvis, match))
46 |             | unvis.set();
47 |     for(int i = 1; i <= nr; ++i) ret[match[i]] = i;

```

```

48 | return ret[0] = 0, ret;
49 | }
50 | // HK, 左到右单向边,  $O(M\sqrt{match})$ 
51 | std::vector<int> e[N];
52 | std::vector<int> matchl, matchr, a, p;
53 | std::vector<int> match(int nl, int nr) {
54 |     matchl.assign(nl + 1, 0), matchr.assign(nr + 1, 0);
55 |     for(;;) {
56 |         a.assign(nl + 1, 0), p.assign(nl + 1, 0);
57 |         static std::queue<int> Q;
58 |         for(int i = 1; i <= nl; ++i)
59 |             if(!matchl[i]) a[i] = p[i] = i, Q.push(i);
60 |         int succ = 0;
61 |         for(; Q.size(); ) {
62 |             int x = Q.front(); Q.pop();
63 |             if(matchl[a[x]]) continue;
64 |             for(int y : e[x]) {
65 |                 if(!matchr[y]) {
66 |                     for(succ = 1; y; x = p[x])
67 |                         | matchr[y] = x, std::swap(matchl[x], y);
68 |                     break;
69 |                 }
70 |                 if(!p[matchr[y]])
71 |                     | Q.push(y = matchr[y]), p[y] = x, a[y] = a[x];
72 |             }
73 |         }
74 |         if(!succ) break;
75 |     }
76 |     return matchl;
77 | } // matchl 是左边每个点匹配的右边点编号
78 | std::pair<std::vector<int>, std::vector<int>> minedgcover(int nl, int nr) {
79 |     match(nl, nr);
80 |     std::vector<int> l, r;
81 |     for(int i = 1; i <= nl; ++i) if(!a[i]) l.push_back(i);
82 |     for(int i = 1; i <= nr; ++i) if(a[matchr[i]]) r.push_back(i);
83 |     return {l, r};
84 | }

```

1.3 网络最大流 | dinic

```

1 | // S 编号最小, T 最大, 或者改一下清空
2 | struct Dinic {
3 |     struct T {
4 |         int to, nxt, v;
5 |         e[N << 3];
6 |         int h[N], head[N], num = 1;
7 |         void link(int x, int y, int v) {
8 |             e[++num] = {y, h[x], v}, h[x] = num;
9 |             e[++num] = {x, h[y], 0}, h[y] = num; // !!!
10 |         }
11 |         int dis[N];
12 |         bool bfs(int s, int t) {
13 |             std::queue<int> q;

```

```

14 |         for(int i = s; i <= t; ++i) dis[i] = -1, head[i] = h[i]; // 如果编号不是
15 |             ↳ [S, T], 只要改这里
16 |             for(Q.push(t), dis[t] = 0; !Q.empty(); ) {
17 |                 | int x = Q.front(); Q.pop();
18 |                 | for(int i = h[x]; i; i = e[i].nxt) if(e[i ^ 1].v && dis[e[i].to] < 0)
19 |                     ↳ {
20 |                         | | dis[e[i].to] = dis[x] + 1, Q.push(e[i].to);
21 |                     }
22 |                 return dis[s] >= 0;
23 |             }
24 |             int dfs(int s, int t, int lim) {
25 |                 if(s == t || !lim) return lim;
26 |                 int ans = 0, mn;
27 |                 for(int & i = head[s]; i; i = e[i].nxt) {
28 |                     | if(dis[e[i].to] + 1 == dis[s] && (mn = dfs(e[i].to, t, std::min(lim,
29 |                         ↳ e[i].v))) {
30 |                         | | e[i].v -= mn, e[i ^ 1].v += mn;
31 |                         | | ans += mn, lim -= mn;
32 |                         | | if(!lim) break;
33 |                     }
34 |                     return ans;
35 |                 }
36 |                 int flow(int s, int t) {
37 |                     int ans = 0;
38 |                     for(; bfs(s, t); ) ans += dfs(s, t, 1e9);
39 |                     return ans;
40 |                 }
41 |             } G;

```

1.4 最小费用流

```

1 | // S 编号最小, T 最大, 或者改一下清空
2 | namespace mcmf {
3 |     using pr = std::pair<l, int>;
4 |     const int N = 10005, M = 1e6 + 10;
5 |     struct edge {
6 |         int to, nxt, v, f;
7 |         e[M << 1];
8 |         int h[N], num = 1;
9 |         void link(int x, int y, int v, int f) {
10 |             e[++num] = {y, h[x], v, f}, h[x] = num;
11 |             e[++num] = {x, h[y], 0, -f}, h[y] = num;
12 |         }
13 |         ll d[N], dis[N];
14 |         int vis[N], fr[N];
15 |         bool spfa(int s, int t) {
16 |             std::queue<int> q;
17 |             std::fill(d + s, d + t + 1, 1e18); // CHECK
18 |             for(d[s] = 0, Q.push(s); !q.empty(); ) {
19 |                 | int x = Q.front(); Q.pop(); vis[x] = 0;
20 |                 | for(int i = h[x]; i; i = e[i].nxt)
21 |                     | | if(e[i].v && d[e[i].to] > d[x] + e[i].f) {

```

```

22 | | | d[e[i].to] = d[x] + e[i].f;
23 | | | fr[e[i].to] = i;
24 | | | if(!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
25 | | | }
26 | | }
27 | | return d[t] < 1e17;
28 | }
29 | bool dijkstra(int s, int t) { // 正常题目不需要 dijk
30 |     std::priority_queue<pr, std::vector<pr>, std::greater<pr>> Q;
31 |     for(int i = s; i <= t; ++i) dis[i] = d[i], d[i] = 1e18, vis[i] = fr[i] =
32 |         ↪ 0; // CHECK
33 |     for(Q.emplace(d[s] = 0, s); !Q.empty(); ) {
34 |         int x = Q.top().second; Q.pop();
35 |         if(vis[x]) continue;
36 |         vis[x] = 1;
37 |         for(int i = h[x]; i; i = e[i].nxt) {
38 |             const ll v = e[i].f + dis[x] - dis[e[i].to];
39 |             if(e[i].v <= d[e[i].to] > d[x] + v) {
40 |                 fr[e[i].to] = i;
41 |                 Q.emplace(d[e[i].to] = d[x] + v, e[i].to);
42 |             }
43 |         }
44 |     }
45 |     for(int i = s; i <= t; ++i) d[i] += dis[i]; // CHECK
46 |     return d[t] < 1e17;
47 | }
48 | std::pair<ll, ll> EK(int s, int t) {
49 |     spfa(s, t); // 如果初始有负权且要 dijk
50 |     ll f = 0, c = 0;
51 |     for(; dijkstra(s, t); ) { // 正常可以用 spfa
52 |         ll fl = 1e18;
53 |         for(int i = fr[t]; i; i = fr[e[i] ^ 1].to) fl = std::min<ll>(e[i].v,
54 |             ↪ fl);
55 |         for(int i = fr[t]; i; i = fr[e[i] ^ 1].to) e[i].v -= fl, e[i] ^ 1.v +=
56 |             ↪ fl;
57 |         f += fl, c += fl * d[t];
58 |         return std::make_pair(f, c);
59 |     }
60 | }

```

```

10 | | | rep(i, 1, nr) vis[i] = 0, mnw[i] = 1e18;
11 | | | while (2333) {
12 | | |     rep(i, 1, nr) if (!vis[i] && chkmin(mnw[i], lw[x] + rw[i] - e[x][i]))
13 | | |         ↪ fa[i] = x;
14 | | |     ll mn = 1e18; int y = -1;
15 | | |     rep(i, 1, nr) if (!vis[i] && chkmin(mn, mnw[i])) y = i;
16 | | |     lw[xx] -= mn; rep(i, 1, nr) if (vis[i]) rw[i] += mn, lw[rpr[i]] -= mn; else
17 | | |         ↪ mnw[i] -= mn;
18 | | |     if (rpr[y]) x = rpr[y], vis[y] = 1; else { while (y
19 | | |         ↪ rpr[y] = fa[y], swap(y, lpr[fa[y]]); return; }
20 | | | }
21 | | | void init(int nl, int nr) {
22 | | |     assert(nl < nr);
23 | | |     KM::nl = nl, KM::nr = nr;
24 | | |     rep(i, 1, nl) lw[i] = -1e18;
25 | | |     rep(i, 1, nl) rep(j, 1, nr) e[i][j] = 0; // or -1e18
26 | | | }
27 | | | void clr() {
28 | | |     rep(i, 1, nl) lpr[i] = lw[i] = 0;
29 | | |     rep(i, 1, nr) rpr[i] = rw[i] = vis[i] = fa[i] = mnw[i] = 0;
30 | | |     rep(i, 1, nl) rep(j, 1, nr) e[i][j] = 0;
31 | | | }
32 | | | void addedge(int x, int y, ll w) { chkmax(e[x][y], w), chkmax(lw[x], w); }
33 | | | ll work() {
34 | | |     rep(i, 1, nl) work(i);
35 | | |     ll tot = 0;
36 | | |     rep(i, 1, nl) tot += e[i][lpr[i]];
37 | | |     return tot;
38 | | | }

```

1.6 一般图最大匹配 | 带花树

```

53 | | | for(int i = fr[t]; i; i = fr[e[i] ^ 1].to) e[i].v -= fl, e[i] ^ 1.v +=
54 | | |     ↪ fl;
55 | | | f += fl, c += fl * d[t];
56 | | }
57 | | return std::make_pair(f, c);
58 | }
59 | // in flow problems with lower bounds (or with negative cycles), flow the
60 |     ↪ negative edges first
    // after the first round, revert the auxiliary edges

```

1.5 二分图最大权匹配 | KM

```

1 | namespace KM {
2 |     int nl, nr;
3 |     ll e[100][100];
4 |     ll lw[100], rw[100];
5 |     int lpr[100], rpr[100];
6 |     int vis[100], fa[100];
7 |     ll mnw[100];
8 |     void work(int x) {
9 |         int xx = x;
10 |
11 |
12 |
13 |
14 |
15 |
16 |
17 |
18 |
19 |
20 |

```

```

21 | }
22 | void aug(int s){
23 |     for (int i=1;i<=n;i++) pre[i]=mk[i]=vis[i]=0,f[i]=1;
24 |     q={};
25 |     mk[s]=1;q.push(s);
26 |     while (q.size()) {
27 |         int x=q.front(); q.pop();
28 |         for (auto v:V[x]) {
29 |             int y=v,z;
30 |             if (mk[y]==2) continue;
31 |             if (mk[y]==1) z=LCA(x,y),flower(x,y,z);
32 |             else if (!match[y]) {
33 |                 for (pre[y]=x;y;) x=pre[y],match[y]=x,swap(y,match[x]);
34 |                 return;
35 |             }
36 |             else pre[y]=x,mk[y]=2,q.push(match[y]),mk[match[y]]=1;
37 |             }
38 |         }
39 |     }
40 |     int work() {
41 |         rep(i,1,n) if (!match[i]) aug(i);
42 |         int res=0;
43 |         rep(i,1,n) res+=match[i]>i;
44 |         return res;
45 |     }
46 | }

```

1.7 最小树形图

抄罗大的，返回值是边的集合，如果没有最小树形图会返回空集（注意 $n = 1$ ），可以修改建图。

```

1 | namespace DMST {
2 |     struct edge {
3 |         int u, v, id; ll w;
4 |         bool operator < (const edge & y) const {
5 |             return w < y.w;
6 |         }
7 |     } ent[N], val[M];
8 |     int ls[M], rs[M], size[M], cc; ll tag[M];
9 |     int fs[N], fw[N], rt[N];
10 |    void put(int x, ll v) {
11 |        if(x) val[x].w += v, tag[x] += v;
12 |    }
13 |    void pushdown(int x) {
14 |        put(ls[x], tag[x]);
15 |        put(rs[x], tag[x]);
16 |        tag[x] = 0;
17 |    }
18 |    int merge(int x, int y) {
19 |        if(!x || !y) return x | y;
20 |        if(val[y] < val[x]) std::swap(x, y);
21 |        pushdown(x), rs[x] = merge(rs[x], y);
22 |        if(size[rs[x]] > size[ls[x]]) {
23 |            std::swap(ls[x], rs[x]);
24 |        }

```

```

25 |         size[x] += size[y];
26 |         return x;
27 |     }
28 |    void ins(int & x, const edge & z) {
29 |        val[+cc] = z, size[cc] = 1;
30 |        x = merge(x, cc);
31 |    }
32 |    void pop(int & x) { x = merge(ls[x], rs[x]); }
33 |    edge top(int x) { return val[x]; }
34 |    int find(int x, int * anc) {
35 |        return anc[x] == x ? x : anc[x] = find(anc[x], anc);
36 |    }
37 |    void link(int u, int v, int w, int id) {
38 |        ins(rt[v], {u, v, id, w});
39 |    }
40 |    int pal[N * 2], tval[N * 2], up[N * 2], end_edge[M], cmt, baned[M];
41 |    std::vector<int> solve(int r) {
42 |        std::queue<int> roots;
43 |        for(int i = 1;i <= n;++i) {
44 |            fs[i] = fw[i] = i, tval[i] = ++ cmt;
45 |            if(i != r) roots.push(i);
46 |        }
47 |        std::vector<edge> H;
48 |        std::vector<int> ret;
49 |        for(;;!roots.empty();){
50 |            int k = roots.front(); roots.pop();
51 |            if(!rt[k]) return ret;
52 |            edge e = top(rt[k]); pop(rt[k]);
53 |            int i = e.u, j = e.v;
54 |            if(find(i, fs) == k) roots.push(k);
55 |            else {
56 |                H.push_back(e); end_edge[e.id] = tval[k];
57 |                if(find(i, fw) != find(j, fw)) {
58 |                    fw[find(j, fw)] = i;
59 |                    ent[k] = e;
60 |                } else {
61 |                    pal[tval[k]] = ++ cmt, up[tval[k]] = e.id;
62 |                    put(rt[k], -e.w);
63 |                    for(;;(e = ent[find(e.u, fs)]).u;){
64 |                        int p = find(e.v, fs);
65 |                        pal[tval[p]] = cmt;
66 |                        up[tval[p]] = e.id;
67 |                        put(rt[p], -e.w);
68 |                        rt[k] = merge(rt[k], rt[p]);
69 |                        fs[p] = k;
70 |                    }
71 |                    tval[k] = cmt;
72 |                    roots.push(k);
73 |                }
74 |            }
75 |        }
76 |        reverse(H.begin(), H.end());
77 |        for(edge i : H) if(!baned[i.id]) {

```

```
78 | | ret.push_back(i.id);
79 | | for(int j = i.v; j != end_edge[i.id]; j = pa[j]) ++ banded[up[j]];
80 | | }
81 | | sort(ret.begin(), ret.end());
82 | | return ret;
83 | | }
84 | }
```

1.8 缩点 | kasaraju

时间复杂度 $O(\frac{n^2}{w})$, 可以对于边修改不多的图快速计算。

```
1 using set = std::bitset<N>;
2 // re 是反向边, 需要连好
3 set e[N], re[N], vis;
4 std::vector<int> sta;
5 void dfs0(int x, set * e) {
6     vis.reset(x);
7     for(;;) {
8         int go = (e[x] & vis)._Find_first();
9         if(go == N) break;
10        | dfs0(go, e);
11    }
12    sta.push_back(x);
13 }
14 std::vector<std::vector<int>> solve() {
15     vis.set();
16     for(int i = 1; i <= n; ++i) if(vis.test(i)) dfs0(i, e);
17     vis.set();
18     auto s = sta;
19     std::vector<std::vector<int>> ret;
20     for(int i = n - 1; i >= 0; --i) if(vis.test(s[i])) {
21         | sta.clear(), dfs0(s[i], re), ret.push_back(sta);
22     }
23     return ret;
24 }
```

1.9 缩点 | Tarjan

```
1 int dfn[sz], low[sz], cc;
2 stack<int> S; int in[sz];
3 int bel[sz], T;
4 void dfs(int x) {
5     dfn[x]=low[x]=++cc; S.push(x), in[x]=1;
6     for (auto v:V[x]) {
7         if (idfn[v]) dfs(v,x), chkmin(low[x], low[v]);
8         | else if (in[v]) chkmin(low[x], dfn[v]);
9     }
10    if (dfn[x]==low[x]) {
11        int y; ++T;
12        | do y=S.top(), S.pop(), in[y]=0, bel[y]=T; while (y!=x);
13    }
14 }
```

1.10 缩点 | 点双

```
1 int dfn[sz], low[sz], cc;
2 stack<int> S;
3 int T;
4 void dfs(int x, int fa) {
5     dfn[x]=low[x]=++cc; S.push(x);
6     for (auto v:V[x]) if (v!=fa) {
7         if (idfn[v]) dfs(v,x), chkmin(low[x], low[v]);
8         | else chkmin(low[x], dfn[v]);
9     }
10    if (fa & low[x] >= dfn[fa]) {
11        int y; ++T;
12        | do y=S.top(), S.pop(), V2[T].push_back(v), V2[y].push_back(T); while (y!
13        | =x);
14        | V2[T].push_back(fa), V2[fa].push_back(T);
15    }
```

1.11 缩点 | 边双

```
1 int dfn[sz], low[sz], cc;
2 stack<int> S;
3 int bel[sz], T;
4 void dfs(int x, int fa) { // cannot handle multiple edges
5     dfn[x]=low[x]=++cc; S.push(x);
6     for (auto v:V[x]) if (v!=fa) {
7         if (idfn[v]) dfs(v,x), chkmin(low[x], low[v]);
8         | else chkmin(low[x], dfn[v]);
9     }
10    if (dfn[x]==low[x]) {
11        int y; ++T;
12        | do y=S.top(), S.pop(), bel[y]=T; while (y!=x);
13    }
14 }
```

1.12 仙人掌

```
1 vector<int> V2[sz], V[sz]; // V2: cactus edges; V: reconstructed tree edges
2 int m; // set to n before dfs
3 void dfs(int x, int f) {
4     static int mark[sz], fa[sz], vis[sz], dep[sz];
5     fa[x]=f; vis[x]=1; dep[x]=dep[f]+1;
6     for (auto v:V2[x]) if (v!=f) {
7         if (i vis[v]) dfs(v,x);
8         | else if (dep[v]<dep[x]) {
9             ++m;
10            | V[v].push_back(m);
11            | for (int y=x; y!=v; y=fa[y]) V[m].push_back(y), mark[y]=1;
12        }
13    }
14    if (!mark[x]) {
15        ++m;
16        | V[fa[x]].push_back(m), V[m].push_back(x);
17    }
18 }
```

1.13 2-Sat

```

1 rep(i,1,n) if (bel[i]<1==bel[i<<1|1]) return puts("IMPOSSIBLE"),0;
2 puts("POSSIBLE");
3 rep(i,1,n) printf("%d ",bel[i<<1]>bel[i<<1|1]);

```

1.14 支配树

```

1 namespace BuildTree {
2     int idom[int>V[int],ANS[int]; // ANS: final tree
3     vector<int>V[int],ANS[int]; // ANS: final tree
4     int deg[int];
5     int fa[int][25],dep[int];
6     int lca(int x,int y) {
7         if (dep[x]<dep[y]) swap(x,y);
8         drep(i,20,0)
9             if (fa[x][i]&&dep[fa[x][i]]>=dep[y])
10                 if (x==y) return x;
11                 else if (x==y) return x;
12                 drep(i,20,0)
13                     if (fa[x][i]!=fa[y][i])
14                         x=fa[x][i],y=fa[y][i];
15                     return fa[x][0];
16             }
17         void work() {
18             queue<int>q;q.push(1);
19             while (!q.empty()) {
20                 int x=q.front();q.pop();
21                 ANS[idom[x]].push_back(x);fa[x][0]=idom[x];dep[x]=dep[idom[x]]+1;
22                 rep(i,1,20) fa[x][i]=fa[fa[x][i-1]][i-1];
23                 for (int v:V[x]) {
24                     --deg[v];if (!deg[v]) q.push(v);
25                     if (!idom[v]) idom[v]=x;
26                     else idom[v]=lca(idom[v],x);
27                 }
28             }
29         }
30     }
31     namespace BuildDAG {
32         vector<int>V[int],rv[int];
33         int dfn[int],id[int],anc[int],cnt;
34         void dfs(int x) {
35             id[dfn[x]]=++cnt;
36             for (int v:V[x]) if (!dfn[v])
37                 BuildTree::V[x].push_back(v), BuildTree::deg[v]++, anc[v] = x,
38                 BuildTree::dfs(v);
39             }
40             int fa[int][25],mn[int];
41             int find(int x) {
42                 if (x==fa[x]) return x;
43                 int tmp=fa[x];fa[x]=find(fa[x]);
44                 chkmin(mn[x],mn[tmp]);
45                 return fa[x];
46             }
47             int semi[int];

```

```

47         void work() {
48             dfs(1);
49             rep(i,1,n) fa[i]=i,mn[i]=1e9,semi[i]=i;
50             drep(w,n,2) {
51                 int x=id[w];int cur=1e9;
52                 if (w>cnt) continue;
53                 for (int v:rv[x]) {
54                     if (!dfn[v]) continue;
55                     if (dfn[v]<dfn[x]) chkmin(cur,dfn[v]);
56                     else find(v),chkmin(cur,mn[v]);
57                 }
58                 semi[x]=id[cur];mn[x]=cur;fa[x]=anc[x];
59                 BuildTree::V[semi[x]].push_back(x);BuildTree::deg[x]++;
60             }
61             void link(int x,int y){V[x].push_back(y),rv[y].push_back(x);}
62         }
63     }

```

1.15 三/四元环

```

1 static int id[int][int],rnk[int];
2 rep(i,1,n) id[i]=i;
3 sort(id+1,id+n+1,[&id][0],return pii{deg[x],x}-pii{deg[y],y});
4 rep(i,1,n) rnk[id[i]]=i;
5 rep(i,1,n) for (auto v:V[i]) if (rnk[v]>rnk[i]) V2[i].push_back(v);
6 int ans3=0; // 3-cycle
7 rep(i,1,n) {
8     static int vis[int];
9     for (auto v:V2[i]) vis[v]=1;
10     for (auto v:V2[i]) for (auto v2:V2[v1]) if (vis[v2]) ++ans3; // (i,v1,v2)
11     for (auto v:V2[i]) vis[v]=0;
12 }
13 ll ans4=0; // 4-cycle
14 rep(i,1,n) {
15     static int vis[int];
16     for (auto v1:V[i]) for (auto v2:V2[v1]) if (rnk[v2]>rnk[i])
17         for (auto v1:V[i]) for (auto v2:V2[v1]) vis[v2]=0;
18 }

```

1.16 双极定向

```

1 vector<int>G[int];
2 namespace bipolar orientation {
3     int dfn[int],low[int],cc,p[int],inv[int],topo[int];
4     bool flg,sgn[int];
5     void dfs(int x,int fa,int s,int t) {
6         dfn[x]=low[x]=++cc; inv[cc]=x,p[x]=fa;
7         if (x==s) dfs(t,x,s,t);
8         for (int y:G[x]) {
9             if (x==s&&y==t) continue;
10             if (!dfn[y]) {
11                 dfs(y,x,s,t);
12                 chkmin(low[x],low[y]);
13                 if (x==s||low[y]>dfn[x]) flg=1;

```



```

14 | | }
15 | | else if (dfn[y]<dfn[x]&&y!=fa) chkmin(low[x],dfn[y]);
16 | | }
17 | }
18 | int check(int s,int t,int n) { // return topo
19 |     if (n==1) return topo[1]=1,1;
20 |     if (s==t) return 0;
21 |     cc=flag=0; dfs(s,s,t);
22 |     if (flag) return 0;
23 |     sgn[s]=0;
24 |     static int pre[sz],suf[sz];
25 |     suf[0]=s,pre[s]=0,suf[s]=t;
26 |     pre[t]=s,suf[t]=n+1,pre[n+1]=t;
27 |     rep(i,3,n) {
28 |         | int v=inv[i];
29 |         | if (!sgn[inv[low[v]]]) {
30 |             | int p=pre[p[v]];
31 |             | pre[v]=p,suf[v]=p[v];
32 |             | suf[p]=pre[p[v]]=v;
33 |         }
34 |         | else {
35 |             | int S=suf[p[v]];
36 |             | pre[v]=p[v],suf[v]=S;
37 |             | suf[p[v]]=pre[S]=v;
38 |         }
39 |         | sgn[p[v]]=!sgn[inv[low[v]]];
40 |     }
41 |     for (int x=s,cnt=0;x!=n+1;x=suf[x]) topo[++cnt]=x;
42 |     return 1;
43 | }
44 | void clr(int n) {
45 |     rep(i,1,n) dfn[i]=low[i]=p[i]=inv[i]=topo[i]=sgn[i]=0,gl[i].clear();
46 | }
47 | }

```

1.17 Tree And Graph

1.17.1 树的计数 Prufer序列

树和其prufer编码——对应，一颗 n 个点的树，其prufer编码长度为 $n-2$ ，且度数为 d_i 的点在prufer编码中出现 d_i-1 次。

由树得到序列：总共需要 $n-2$ 步，第 i 步在当前的树中寻找具有最小标号的叶子节点，将其相连的点的标号设为Prufer序列的第 i 个元素 p_i ，并将此叶子节点从树中删除，直到最后得到一个长度为 $n-2$ 的Prufer序列和一个只有两个节点的树。

由序列得到树：先将所有点的度赋初值为 1，然后加上它的编号在Prufer序列中出现的次数，得到每个点的度；执行 $n-2$ 步，第 i 步选取具有最小标号的度为 1 的点 u 与 $v = p_i$ 相连，得到树中的一条边，并将 u 和 v 的度减一。最后再把剩下的两个度为 1 的点连边，加入到树中。

相关结论： n 个点完全图，每个点度数依次为 d_1, d_2, \dots, d_n ，这样生成树的棵数为：

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}.$$

左边有 n_1 个点，右边有 n_2 个点的完全二分图的生成树棵数为 $n_1^{n_2-1} \times n_2^{n_1-1}$ 。

m 个连通块，每个连通块有 c_i 个点，把他们全部连通的生成树方案数： $(\sum c_i)^{m-2} \prod c_i$

1.17.2 有根树的计数

首先，令 $S_{n,j} = \sum_{1 \leq i \leq n/j}$ ；于是 $n+1$ 个结点的有根树的总数为 $a_{n+1} = \frac{\sum_{j=1}^n j a_j S_{n-j}}{n}$ 。注： $a_1 = 1, a_2 = 1, a_3 = 2, a_4 = 4, a_5 = 9, a_6 = 20, a_9 = 286, a_{11} = 1842$ 。

1.17.3 无根树的计数

n 是奇数时，有 $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$ 种不同的无根树。

n 是偶数时，有 $a_n - \sum_{i=1}^{n/2} a_i a_{n-i} + \frac{1}{2} a_{n/2} (a_{n/2} + 1)$ 种不同的无根树。

1.17.4 生成树计数 Kirchhoff's Matrix-Tree Theorem

Kirchoff Matrix $T = Deg - A$, Deg 是度数对角阵， A 是邻接矩阵。无向图度数矩阵是每个点度数；有向图度数矩阵是每个点入度。

邻接矩阵 $A[u][v]$ 表示 $u \rightarrow v$ 边个数，重边按照边数计算，自环不计入度数。

无向图生成树计数： $c = |K|$ 的任意1个 $n1$ 阶主子式 |

有向图外向树计数： $c = |$ 去掉根所在的那阶得到的主子式 |

1.17.5 有向图欧拉回路计数 BEST Theorem

$$ec(G) = t_w(G) \prod_{v \in V} (\deg(v) - 1)!$$

其中 \deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数。相关计算参考生成树计数。

欧拉连通图中任意两点外向树个数相同: $t_v(G) = t_w(G)$ 。

以 1 结尾的欧拉路径计数就是把 \deg 视为出度，把 $\deg(1)$ 的贡献改为 $\deg(1)!$ 。

1.17.6 Tutte Matrix

Tutte matrix A of a graph $G = (V, E)$:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (i, j) \in E \text{ and } i < j \\ -x_{ij} & \text{if } (i, j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij} , $i < j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

1.17.7 Edmonds Matrix

Edmonds matrix A of a balanced $(|U| = |V|)$ bipartite graph $G = (U, V, E)$:

$$A_{ij} = \begin{cases} x_{ij} & (u_i, v_j) \in E \\ 0 & (u_i, v_j) \notin E \end{cases}$$

where the x_{ij} are indeterminates. G 有完美匹配当且仅当关于 x_{ij} 的多项式 $\det(A_{ij})$ 不为 0. 完美匹配个数等于多项式中单项式的个数。

1.18 拟阵交

```

1 // max size, minimum weight
2 namespace MatroidIntersection {
3     int K;
4     ll w[sz]; // weight
5     int in[sz]; // ans
6     namespace Check { // implementation needed

```

```

7 // recommend writing checker here
8 void init() {}
9 // return {-1} if no cycle; return cycle otherwise.
10 vector<int> cycleA(int x) {}
11 vector<int> cycleB(int x) {}
12 // not necessary
13 void check() {init();}
14 }
15 bool work() { // try augment
16     using pli=pair<ll, int>;
17     static vector<int> V[1sz];
18     static pli fr[1sz];
19     Check::init();
20     rep(i,1,K) V[i].clear();
21     vector<int> A,B;
22     rep(i,1,K) if (!in[i]) {
23         auto cyca=Check::cycleA(i);
24         if (cyca.size()==1u68cyca[0]==-1) A.push_back(i);
25         else for (auto y:cyca) V[y].push_back(i);
26         auto cycb=Check::cycleB(i);
27         if (cycb.size()==1u68cycb[0]==-1) B.push_back(i);
28         else for (auto y:cycb) V[i].push_back(y);
29     }
30     rep(i,1,K) dis[i]={ll(1e18),K+1},fr[i]=0;
31     priority_queue<pair<pli,int>, vector<pair<pli,int>>,
32         ↳greater<pair<pli,int>>>q;
33     for (auto x:A) dis[x]={W[x],0},q.push({dis[x],x});
34     while (!q.empty()) {
35         auto [mw,x]=q.top(); q.pop();
36         if (dis[x]!=mw) continue;
37         for (auto v:V[x])
38             if (chkmin(dis[v],{dis[x].fir+W[v],dis[x].sec+1}))
39                 q.push({dis[v],v}),fr[v]=x;
40     }
41     pli mn={ll(1e18),K+1}; int mnp=-1;
42     for (auto x:B) if (chkmin(mn,dis[x])) mnp=x;
43     if (mnp==-1) return 0;
44     for (int x=mnp;x=fr[x]) in[x]^=1;
45     Check::check();
46     return 1;
47 }
48 void clr() {
49     rep(1,1,K) in[i]=0;
50 }
51 }

```

2 数论

2.1 取模还原分数

```

1 std::pair<int, int> approx(int p, int q, int A) {
2     | int x = q, y = p, a = 1, b = 0;
3     | for(;;x > A;) {

```

```

4 | | std::swap(x, y), std::swap(a, b);
5 | | a -= x / y * b, x %= y;
6 | | }
7 | | return {x, a};
8 | // q ≡  $\frac{x}{a}$  (mod p), x ≤ A, |a| 取到最小值

```

2.2 扩展欧几里得

```

1 // result : -b < x < b AND -a < y <= a when a,b != 0
2 void exgcd(ll a, ll b, ll &x, ll &y) {
3     | if(!b) return x = 1, y = 0, void();
4     | exgcd(b, a % b, y, x), y -= a / b * x;
5 }

```

2.3 万能欧几里得

```

1 // 万欧
2 // 前提 : r < q, r >= q 先提几个 U 出来再用
3 // 使用: y * q <= X * p + r, 斜率 p/q, U表示向上, R表示到达一个顶点, 先一些 U 再一个 R
4 template<class T>
5 T power(T a, ll k) {
6     | // 有效率需求可以改为半群乘法
7     | if(!k) return T();
8     | T res = a;
9     | for(;;k>0) {
10        | | if(k & 1) res = res * a;
11        | | if(k >= 1) a = a * a;
12        | | }
13        | return res;
14    }
15    template<class T>
16    T solve(ll p, ll q, ll r, ll l, T U, T R) {
17        | if (p >= q)
18        | | return solve(p % q, q, r, l, U, power(U, p / q) + R);
19        | ll m = ((-int128)p * l + r) / q;
20        | if (im) return power(R, l);
21        | ll cnt = l - ((-int128)q * m - r - 1) / p;
22        | return power(R, (q - r - 1) / p) + U + solve(q, p, (q - r - 1) % p, m - 1,
23        | ↳R, U) + power(R, cnt);
24    }

```

2.4 直线下点数|欧几里得

$n < 2^{32}, 1 \leq m < 2^{32}$

$$result = \sum_{i=0}^{n-1} \lfloor \frac{ai+b}{m} \rfloor \pmod{2^{63}}$$

```

1 u64 floor_sum(u64 n, u64 m, u64 a, u64 b) {
2     | u64 ans = 0;
3     | for(;;) {
4         | if(a >= m) ans += n * (n - 1) / 2 * (a / m), a %= m;
5         | if(b >= m) ans += n * (b / m), b %= m;
6         | u64 ymax = a * n + b; // use u128 if it's big

```

```
7 | | if(ymax < m) break;
8 | | n = ymax / m;
9 | | b = ymax % m;
10 | | std::swap(m, a);
11 | }
12 | return ans;
13 | }
```

2.5 Stern-Brocot Tree 二分

```
1 using cp = std::complex<ll>;
2 cp fracBS(ll n, ll m, auto f) {
3 |     bool dir = 1, A = 1, B = 1;
4 |     cp lo(0, 1), hi(1, 1); // hi can be (1, 0), f(hi) must be true
5 |     if (f(lo)) return lo;
6 |     while(A || B) {
7 |         ll adv = 0, s = 1;
8 |         for (int x = 0; s(s * 2) >= x) {
9 |             | adv += s;
10 |             | cp mid = lo * adv + hi;
11 |             | if (mid.real() > n || mid.imag() > m || dir == !f(mid)) {
12 |                 | adv -= s, x = 2;
13 |             | }
14 |             | }
15 |             | hi += lo * adv, dir = !dir;
16 |             | swap(lo, hi);
17 |             | A = B, B = adv;
18 |         }
19 |         return dir ? hi : lo;
20 |     } // 返回值是最小的使得 f 为真的
21 |     // 另外一个是最大的使得 f 为假的
```

2.6 扩展中国剩余定理

```
1 ll exCRT(ll a1, ll p1, ll a2, ll p2) {
2 |     ll a, b, gcd = std::gcd(p1, p2);
3 |     if((a1 - a2) % gcd)
4 |         | return -1;
5 |     exgcd(p1, p2, a, b);
6 |     ll k = i128((a2 - a1) % p2 + p2) * (a + p2) % p2;
7 |     return p1 / gcd * k + a1;
8 | }
```

2.7 Miller-Rabin

```
1 using f64 = long double;
2 ll p;
3 f64 invp;
4 void setmod(ll x) {
5 |     p = x, invp = (f64) 1 / x;
6 | }
7 ll mul(ll a, ll b) {
8 |     ll z = a * invp * b + 0.5;
9 |     ll res = a * b - z * p;
10 |     return res + (res >> 63 & p);
11 | }
```

```
12 ll pow(ll a, ll x, ll res = 1) {
13 |     for(;x;x>= 1, a = mul(a, a))
14 |         | if(x & 1) res = mul(res, a);
15 |     return res;
16 | }
17 bool checkprime(ll p) {
18 |     if(p == 1) return 0;
19 |     setmod(p);
20 |     ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
21 |     for(ll a : {2, 3, 5, 7, 11, 13, 82, 373}) {
22 |         | if(a % p == 0)
23 |             | continue;
24 |         | ll x = pow(a, s), y;
25 |         | for(int i = 0; i < d; ++i, x = y) {
26 |             | y = mul(x, x);
27 |             | if(y == 1 && x != 1 && x != p - 1)
28 |                 | return 0;
29 |         | }
30 |         | if(x != 1) return 0;
31 |     }
32 |     return 1;
33 | }
```

2.8 Pollard-rho

```
1 ll rho(ll n) {
2 |     if(!n & 1) return 2;
3 |     static std::mt19937_64 gen((size_t)"hehezhou");
4 |     ll x = 0, y = 0, prod = 1;
5 |     auto f = [&](ll o) { return mul(o, o) + 1; };
6 |     setmod(n);
7 |     for(int t = 30, z = 0; t % 64 || std::gcd(prod, n) == 1; ++t) {
8 |         | if (x == y) x = ++z, y = f(x);
9 |         | if(!ll q = mul(prod, x + n - y)) prod = q;
10 |         | x = f(x), y = f(f(y));
11 |     }
12 |     return std::gcd(prod, n);
13 | }
14 std::vector<ll> factor(ll x) {
15 |     std::vector<ll> res;
16 |     auto f = [&](auto f, ll x) {
17 |         | if(x == 1) return ;
18 |         | if(checkprime(x)) return res.push_back(x);
19 |         | ll y = rho(x);
20 |         | f(f, y), f(f, x / y);
21 |     };
22 |     f(f, x), sort(res.begin(), res.end());
23 |     return res;
24 | }
```

3 Math

3.1 拉格朗日反演

$$G(F(x)) = H(x) \Rightarrow [x^n]G(x) = \frac{1}{n} [u^{n-1}]H'(u) \left(\frac{u}{F(u)} \right)^n$$

$$G(F(x)) = x \Rightarrow [x^n]H(G(x)) = \frac{1}{n} [u^{n-1}]H'(u) \left(\frac{u}{F(u)} \right)^n$$

$$G(F(x)) = x \Rightarrow [x^n]G^k(x) = \frac{k}{n} [u^{n-k}] \left(\frac{u}{F(u)} \right)^n$$

3.2 分拆数/五边形数

$$\prod_{i \geq 1} (1 - x^i) = \sum_{k=-\infty}^{\infty} (-1)^k x^{\frac{k(3k-1)}{2}}$$

3.3 Fast Fourier Transform

```

1 using db = double;
2 using cp = std::complex<db>;
3 // cp::real, cp::imag, std::conj, std::arg
4 const db pi = std::acos(-1);
5 int rev[N], lim;
6 cp wn[N];
7
8 void init(int len) {
9     lim = 2 << std::lg(len - 1);
10    for(static int i = 1; i < lim; i += i) {
11        for(int j = 0; j < i; ++j) {
12            | wn[i + j] = std::polar(1., db(j) / i * pi);
13        }
14    }
15    for(int i = 1; i < lim; ++i) {
16        | rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
17    }
18 }
19
20 void DFT(cp * a) {
21     for(int i = 0; i < lim; ++i) {
22         | if(rev[i] < i) std::swap(a[rev[i]], a[i]);
23     }
24     for(int i = 1; i < lim; i += i) {
25         for(int j = 0; j < lim; j += i + i) {
26             | for(int k = 0; k < i; ++k) {
27                 | cp x = a[i + j + k] * wn[i + k];
28                 | a[i + j + k] = a[k + j] - x;
29                 | a[k + j] += x;
30             }
31         }
32     }
33 }
34
35 void IDFT(cp * a) {
36     DFT(a), std::reverse(a + 1, a + lim);
37     for(int i = 0; i < lim; ++i)
38         | a[i] /= lim;
39 }
40

```

3.4 Number Theoretic Transform

```

1 int rev[N], wn[N], lim, invlim;
2 int pow(int a, int b, int ans = 1) {
3     | for(; b >= 1, a = (u64) a * a % mod) if(b & 1)
4         | ans = (u64) ans * a % mod;
5     | return ans;
6 }
7
8 void init(int len) {
9     | lim = 2 << std::lg(len - 1);
10    | invlim = mod - (mod - 1) / lim;
11    for(static int i = 1; i < lim; i += i) {
12        | wn[i] = 1;
13        | const int w = pow(3, mod / i / 2);
14        | for(int j = 1; j < i; ++j) {
15            | wn[i + j] = (u64) wn[i + j - 1] * w % mod;
16        }
17    }
18    for(int i = 1; i < lim; ++i) {
19        | rev[i] = rev[i >> 1] >> 1 | (i % 2u * lim / 2);
20    }
21 }
22
23 void DFT(int * a) {
24     static u64 t[N];
25     for(int i = 0; i < lim; ++i) t[i] = a[rev[i]];
26     for(int i = 1; i < lim; i += i) {
27         for(int k = i & (1 << 19); k--;)
28             | if(t[k] >= mod * 9uLL) t[k] -= mod * 9uLL;
29         for(int j = 0; j < lim; j += i + i) {
30             | for(int k = 0; k < i; ++k) {
31                 | const u64 x = t[i + j + k] * wn[i + k] % mod;
32                 | t[i + j + k] = t[k + j] + (mod - x), t[k + j] += x;
33             }
34         }
35     }
36     for(int i = 0; i < lim; ++i) a[i] = t[i] % mod;
37 }
38
39 void IDFT(int * a) {
40     DFT(a), std::reverse(a + 1, a + lim);
41     for(int i = 0; i < lim; ++i)
42         | a[i] = (u64) a[i] * invlim % mod;
43 }
44

```

3.5 Generating function

```

1 void cpy(int * a, int * b, int n) {
2     | if(a != b) memcpy(a, b, n << 2);
3     | memset(a + n, 0, (lim - n) << 2);
4 }
5
6 void inv(int * a, int * b, int n) { // b = inv(a) mod x^n
7     | if(n == 1) return void(*b = pow(*a, mod - 2));
8     static int c[N], d[N];
9     | int m = (n + 1) / 2;
10    | inv(a, b, m);
11    | init(n + m), cpy(c, b, m), cpy(d, a, n);

```

```

11 | DFT(c), DFT(d);
12 | for(int i = 0; i < lim; ++i) c[i] = (u64) c[i] * c[i] % mod * d[i] % mod;
13 | IDFT(c);
14 | for(int i = m; i < n; ++i) b[i] = norm(mod - c[i]);
15 | }
16 | void log(int * a, int * b, int n) { // b = log(a) (mod x^n)
17 |     static int c[N], d[N];
18 |     inv(a, c, n), init(n + n);
19 |     for(int i = 1; i < n; ++i) d[i - 1] = (u64) a[i] * i % mod;
20 |     cpy(d, d, n - 1), cpy(c, c, n);
21 |     DFT(c), DFT(d);
22 |     for(int i = 0; i < lim; ++i) c[i] = (u64) c[i] * d[i] % mod;
23 |     IDFT(c), *b = 0;
24 |     for(int i = 1; i < n; ++i) b[i] = pow(i, mod - 2, c[i - 1]);
25 | }

```

3.6 全在线卷积

```

1 | struct oc {
2 |     std::vector<int> f, g, res;
3 |     std::vector<std::vector<int>> fa, fb;
4 |     int n, p;
5 |     oc(int n) : f(n), g(n), res(n), n(n), p(0) { }
6 |     void push(int v0, int v1) {
7 |         f[p] = v0;
8 |         res[p] = (res[p] + (u64) f[0] * v1 + (u64) g[0] * v0) % mod;
9 |         g[p++] = v1;
10 |         static int A[N], B[N];
11 |         int lb = p & -p;
12 |         init(lb * 2);
13 |         memset(A, 0, lim << 2);
14 |         memset(B, 0, lim << 2);
15 |         for(int i = 0; i < lb; ++i) A[i] = g[p - lb + i], B[i] = f[p - lb + i];
16 |         DFT(A), DFT(B);
17 |         if(lb == p) {
18 |             fa.emplace_back(A, A + lim);
19 |             fb.emplace_back(B, B + lim);
20 |             for(int i = 0; i < lim; ++i) A[i] = (u64) A[i] * B[i] % mod;
21 |         } else {
22 |             auto & C = fb[std::min(g(lim)], & D = fa[std::min(g(lim))];
23 |             for(int i = 0; i < lim; ++i) A[i] = ((u64) A[i] * C[i * 2] + (u64)
24 |                 ↪ B[i] * D[i * 2]) % mod;
25 |         }
26 |         IDFT(A);
27 |         for(int j = p; j < p + lb && j < n; ++j) res[j] = (res[j] + A[j - p +
28 |             ↪ lb]) % mod;
29 |     }
30 | };
31 | struct Exp : oc {
32 |     std::vector<int> res;
33 |     Exp(int n) : oc(n), res(n) { }
34 |     void push(int v) {
35 |         if(!res[0]) return void(res[0] = 1);
36 |         oc::push(res[p], v * u64(p + 1) % mod);
37 |     }
38 | };

```

```

35 |         res[p] = (u64) oc::res[p - 1] * inv[p] % mod;
36 |     }
37 | };
38 | struct ln : oc {
39 |     std::vector<int> res; int fi;
40 |     ln(int n) : oc(n), res(n), fi(0) {}
41 |     void push(int v) {
42 |         if(!fi) return void(fi = 1);
43 |         oc::push(res[p] * (u64) p % mod, v);
44 |         res[p] = ((u64) v * p + mod - oc::res[p - 1]) % mod * inv[p] % mod;
45 |     }
46 | };
47 | struct Inv : oc {
48 |     std::vector<int> res; int fi;
49 |     Inv(int n) : oc(n), res(n), fi(0) {}
50 |     void push(int v) {
51 |         res[p] = fi ? (oc::res[p] + (u64) v * res[0]) % mod * (mod - res[0]) %
52 |             ↪ mod : pow(fi = v, mod - 2);
53 |         oc::push(res[p], v);
54 |     }
55 | };

```

3.7 Berlekamp Massey

```

1 | vector<int> berlekamp_massey(const vector<int> &a) {
2 |     vector<int> v, last; // v is the answer, 0-based
3 |     int k = -1, delta = 0;
4 |     for (int i = 0; i < (int)a.size(); i++) {
5 |         int tmp = 0;
6 |         for (int j = 0; j < (int)v.size(); j++)
7 |             tmp = (tmp + (long long)a[i - j - 1] * v[j]) % p;
8 |         if (a[i] == tmp) continue;
9 |         if (k < 0) {
10 |             k = i; delta = (a[i] - tmp + p) % p;
11 |             v = vector<int>(i + 1); continue; }
12 |         vector<int> u = v;
13 |         int val = (long long)(a[i] - tmp + p) *
14 |             qpow(delta, p - 2) % p;
15 |         if (v.size() < last.size()) + i - k)
16 |             v.resize(last.size() + i - k);
17 |         (v[i - k - 1] += val) %= p;
18 |         for (int j = 0; j < (int)last.size(); j++) {
19 |             v[i - k + j] = (v[i - k + j] -
20 |                 ↪ (long long)val * last[j]) % p;
21 |             if (v[i - k + j] < 0) v[i - k + j] += p; }
22 |         if ((int)u.size() - i < (int)last.size() - k) {
23 |             last = u; k = i; delta = a[i] - tmp;
24 |             if (delta < 0) delta += p; } }
25 |     for (auto &x : v) x = (p - x) % p;
26 |     v.insert(v.begin(), 1); //一般是需要最小递推式的，处理一下
27 |     return v; }
28 | //  $\forall i, \sum_{j=0}^m a_i - p_j = 0$ 

```

3.8 线性规划 | 单纯形法

```
1 using db = long double;
2 const db eps = 1e-16;
3 int sgn(db x) { return x < -eps ? -1 : x > eps; }
4 namespace LP {
5     const int N = 21, M = 21;
6     int n, m; // n : 变量个数, m : 约束个数
7     db a[M + N][N], x[N + M];
8     // 约束: 对于 1 <= i <= m : a[i][0] + \sum_j x[j] * a[i][j] >= 0
9     // x[j] >= 0
10    // 最大化 \sum_j x[j] * a[0][j]
11    int id[N + M];
12    void pivot(int p, int o) {
13        std::swap(id[p], id[o + n]);
14        db w = -a[o][p];
15        for(int i = 0; i <= n; ++i) a[o][i] /= w;
16        a[o][p] = -1 / w;
17        for(int i = 0; i <= m; ++i) if(sgn(a[i][p]) && i != o) {
18            db w = a[i][p]; a[i][p] = 0;
19            for(int j = 0; j <= n; ++j) a[i][j] += w * a[o][j];
20        }
21    }
22    db solve() { // nan : 无解, inf : 无界, 否则返回最大值
23        for(int i = 1; i <= n + m; ++i) id[i] = i;
24        for(;;) {
25            int p = 0, min = 1;
26            for(int i = 1; i <= m; ++i) {
27                if(a[i][0] < a[min][0]) min = i;
28            }
29            if(a[min][0] >= -eps) break;
30            for(int i = 1; i <= n; ++i) if(a[min][i] > eps && id[i] > id[p]) {
31                p = i;
32            }
33            if(!p) return nan("");
34            pivot(p, min);
35        }
36        for(;;) {
37            int p = 1;
38            for(int i = 1; i <= n; ++i) if(a[0][i] > a[0][p]) p = i;
39            if(a[0][p] < eps) break;
40            db min = INFINITY; int o = 0;
41            for(int i = 1; i <= m; ++i) if(a[i][p] < -eps) {
42                db w = -a[i][0] / a[i][p]; int d = sgn(w - min);
43                if(d < 0 || id && id[i] > id[o]) o = i, min = w;
44            }
45            if(!o) return INFINITY;
46            pivot(p, o);
47        }
48        for(int i = 1; i <= m; ++i) x[id[i] + n] = a[i][0];
49        return a[0][0];
50    }
51 }
```

3.9 Simpson 积分

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}(f(x_0) + 4 \sum_{i=1}^{n/2} f(x_{2i-1}) + 2 \sum_{i=1}^{n/2-1} f(x_{2i}) + f(x_n))$$
$$\approx \frac{3(b-a)}{8n}(f(x_0) + 3 \sum_{i=1}^{n/3} (f(x_{3i-1}) + f(x_{3i-2})) + 2 \sum_{i=1}^{n/3-1} f(x_{3i}) + f(x_n))$$

3.10 黄金三分

```
1 db findmax(db a, db c, auto f) {
2     auto g = [ε](db l, db r) {
3         return l + (r - l) * (std::numbers::phi_v<db> - 1);
4     };
5     db b = g(a, c), bv = f(b);
6     for(int i = 0; i < 45; ++i) {
7         db x = g(a, b), xv = f(x);
8         if(xv > bv) { // change here if findmin
9             c = b, b = x, bv = xv;
10        } else {
11            a = c, c = x;
12        }
13    }
14    return bv;
15 } // log_{1.618}(2) ≈ 1.44
```

4 字符串

4.1 后缀自动机 | SAM

需要两倍点数量。

```
1 int ch[N][26], lk[N], len[N], nd = 1, las = 1;
2 void extend(int c, int k) {
3     int x = ++ nd, p = las; las = x;
4     len[x] = len[p] + 1;
5     for(;; p && !ch[p][c]; p = lk[p]) ch[p][c] = x;
6     if(!p) return lk[x] = 1, void();
7     int q = ch[p][c];
8     if(len[q] == len[p] + 1)
9         return lk[x] = q, void();
10    int cl = ++ nd;
11    len[cl] = len[p] + 1;
12    memcpy(ch[cl], ch[q], 104);
13    lk[cl] = lk[q], lk[q] = lk[x] = cl;
14    for(;; p && ch[p][c] == q; p = lk[p]) ch[p][c] = cl;
15    void init() {
16        static int bin[N];
17        memset(bin, 0, sizeof (int)) * (n + 1);
18        for(int i = 1; i <= nd; i++) ++ bin[len[i]];
19        for(int i = 1; i <= n; i++) bin[i] += bin[i - 1];
20        for(int i = nd; i; i--) A[bin[len[i]]--] = i;
21    }
```

4.2 基本子串字典

```

1  for(int i = 2; i <= T[0].nd; i++) {
2      | int x = T[0].A[i];
3      | int R = T[0].r[x], L = R - T[0].len[x] + 1;
4      | int y = T[1].fnd(T[1].ed[L], R - L + 1);
5      | if(T[1].len[y] == R - L + 1) {
6          | ++cnt; T[0].tag[x] = cnt; T[1].tag[y] = cnt;
7          | rt[0][cnt] = x, rt[1][cnt] = y;
8      | }
9  }
10 for(int o = 0; o < 2; o++)
11     for(int i = T[o].nd; i > 1; i--) {
12         | int x = T[o].A[i];
13         | if(T[o].tag[x]) continue;
14         | for(int k = 0; k < 26; k++)
15             | if(T[o].ch[x][k]) T[o].tag[x] = T[o].tag[T[o].ch[x][k]];
16     }
17     for(int o = 0; o < 2; o++)
18         for(int i = 2; i <= T[o].nd; i++) {
19             | int x = T[o].A[i];
20             | vec[o][T[o].tag[x]].pb(x);
21         } // vec[0] : from left to right, node id of the column , vec[1] : from down
22             ↳ to up
23         ↳ u : T[0].r[rt] - T[0].len[rt] + 1, D = U + vec[1][t].size() - 1, L = R -
24             ↳ vec[0][t].size() + 1, R = T[0].r[rt]
25         | int x = T[0].fnd(T[0].ed[r], r - l + 1);
26         | int blk = T[0].tag[x];
27         ↳ distance to the right , 0 - base
28         | int rp = T[0].r[rt][blk] - T[0].r[x];
29         ↳ distance to the up // the upper - right point is (T[0].r[rt] -
30             ↳ T[0].len[rt] + 1, T[0].r[rt])
31         | int lp = T[0].r[x] - (r - l) - (T[0].r[rt][blk] - T[0].len[rt][blk] +
32             ↳ 1);

```

4.3 DAG 剖分

```

1  void build() {
2      | for(int i = 2; i <= nd; i++)
3          | e[lk[i]].pb(i);
4      | static int q[N], d[N];
5      | for(int i = 1; i <= nd; i++)
6          | for(int j = 0; j < 26; j++)
7              | if(ch[i][j]) ++d[ch[i][j]];
8      | int hd = 1, tl = 0;
9      | q[tl] = 1;
10     | while(hd <= tl) {
11         | int x = q[hd++];
12         | for(int i = 0; i < 26; i++)
13             | if(ch[x][i]) {
14                 | int v = ch[x][i];
15                 | if((--d[v]) == 0) q[tl++] = v;
16             | }
17     }
18     | static ll f[N], h[N];

```

```

19     | for(int i = tl, x; i; i--) {
20         | f[x = q[i]]++;
21         | for(int j = 0; j < 26; j++)
22             | if(ch[x][j]) f[x] += f[ch[x][j]];
23     }
24     | for(int i = 1, x; i <= tl; i++) {
25         | h[x = q[i]]++;
26         | for(int j = 0; j < 26; j++)
27             | if(ch[x][j]) h[ch[x][j]] += h[x];
28     }
29     | static int nx[N], fr[N];
30     | for(int i = 1; i <= nd; i++) {
31         | for(int j = 0; j < 26; j++)
32             | if(ch[i][j] && f[ch[i][j]] > f[nx[i]]) nx[i] = ch[i][j];
33         | for(int j = 0; j < 26; j++)
34             | if(ch[i][j] && h[i] > h[fr[ch[i][j]]]) fr[ch[i][j]] = i;
35     }
36     | fr[0] = nx[0] = 0;
37     | static bool vis[N];
38     | for(int i = 1; i <= nd; i++) {
39         | if(fr[nx[i]] == i) son[i] = nx[i], vis[son[i]] = 1;
40     }
41 }

```

4.4 exKMP

```

1  static int lcp[N];
2  | int mx=1, pt=1; lcp[1]=n;
3  | for(int i=2; i<=n; i++){
4      | if(i<=mx) lcp[i]=min(lcp[i-pt+1],mx-i+1);
5      | while(i+lcp[i]<=n && S[i+lcp[i]]==S[1+lcp[i]]) ++lcp[i];
6      | if(i+lcp[i]-1>mx) pt=i, mx=i+lcp[i]-1;
7  }

```

4.5 log 个最小后缀

```

1  for(int i = 1; i <= n; i++) {
2      | St.pb(i); vector<int> nw;
3      | for(auto t : St) {
4          | bool ok = true;
5          | while(inw.empty()){
6              | int x = nw.back();
7              | if(S[i] > S[i-t+x]) ok = false;
8              | if(S[i] >= S[i-t+x]) break; nw.pop_back();
9          | }
10         | if(ok && (nw.empty() || (i - t + 1 <= t - nw.back()))) nw.pb(t);
11         | St = nw;
12     }
13     | for(int x : St){
14         | bool FLAG = true;
15         | while(nx.size()){
16             | int y = nx.back(); int lcp = LCP(x, y); if(x + lcp - 1 >= r) break;
17             | if(S[x + lcp] > S[y + lcp]){ FLAG = false; break; } nx.pop_back();
18             | if(FLAG && (nx.empty() || r - x + 1 <= x - nx.back())) nx.pb(x);
19         } // in segmentree, work(L, ans, rpos), work(R, ans, rpos), then return ans

```


4.6 SA

```
1 char s[N]; int m, rk[N * 2], sa[N], tmp[N * 2], h[N], y[N];
2 void Sort() {
3     for(int i=1; i<=m; i++) c[i] = 0;
4     for(int i=1; i<=n; i++) c[rk[i]]++;
5     for(int i=1; i<=m; i++) c[i] += c[i-1];
6     for(int i=n; i>=1; i--) sa[c[rk[i]]--] = y[i];
7 }
8 void get_sa() {
9     for(int i=1; i<=n; i++) rk[i] = s[i], y[i] = i; Sort();
10    for(int k=1; k<=n; k<=1) {
11        int ret = 0;
12        for(int i=n-k+1; i<=n; i++) y[++ret] = i;
13        for(int i=1; i<=n; i++) if(sa[i] > k) y[++ret] = sa[i] - k;
14        Sort();
15        for(int i = 1; i <= n; i++) swap(rk[i], tmp[i]);
16        rk[sa[1]] = 1; int num = 1;
17        for(int i=2; i<=n; i++) {
18            if(tmp[sa[i]] == tmp[sa[i-1]] && tmp[sa[i]+k] == tmp[sa[i-1]+k])
19                rk[sa[i]] = num;
20            else rk[sa[i]] = ++num;
21            } m = num;
22    }
23 }
24 void get_h() {
25     int k = 0;
26     for(int i=1; i<=n; i++) {
27         if(rk[i]==1) continue;
28         int j = sa[rk[i]-1]; if(k) k--;
29         while(i+k<=n && j+k<=n && s[i+k]==s[j+k]) k++;
30         h[rk[i]] = k;
31     }
32 }
```

4.7 PAM

```
1 namespace pam {
2     int ch[N][26], len[N], lk[N], rp, las, nd, top[N], d[N];
3     void init() { rp = 0, las = nd = 1, len[1] = -1, lk[0] = 1; }
4     // remember to set S[0] = *
5     int jmp(int x) { while(S[rp - len[x] - 1] != S[rp]) x = lk[x]; return x; }
6     void ins(int c) {
7         ++rp; int p = jmp(las);
8         if(!ch[p][c]) {
9             int x = ++nd;
10            len[x] = len[p] + 2;
11            lk[x] = ch[jmp(lk[p])][c];
12            ch[p][c] = x;
13            if(len[x] - len[lk[x]] == d[lk[x]])
14                top[x] = top[lk[x]], d[x] = d[lk[x]];
15            else {
16                top[x] = x;
17                d[x] = len[x] - len[lk[x]];
18            }
19 }
```

```
19 | } las = ch[p][c];
20 | }
21 | }
22 while(x) {
23     if(pam :: d[x] == pam :: d[pam :: lk[x]]) {
24         // when doing dp, the position i - len[x] ~
25         // i - (len[top[x]] + d[x]) have been updated (in i - d[x])
26         g[x] = f[i - pam :: len[pam :: top[x]]];
27         Add(g[x], g[pam :: lk[x]]);
28     }
29     else {
30         // update from i - len[x]
31         g[x] = f[i - pam :: len[x]];
32     }
33     Add(f[i], g[x]);
34     x = pam :: lk[pam :: top[x]];
35 }
```

4.8 AC 自动机

```
1 void init() {
2     queue <int> q;
3     for(int i = 0; i < 26; i++)
4         if(ch[0][i]) q.push(ch[0][i]);
5     while(!q.empty()) {
6         int x = q.front(); q.pop();
7         e[lk[x]].pb(x);
8         for(int i = 0; i < 26; i++) {
9             if(ch[x][i]) {
10                 lk[ch[x][i]] = ch[lk[x]][i];
11                 q.push(ch[x][i]);
12             }
13             else ch[x][i] = ch[lk[x]][i];
14         }
15     }
16 }
```

4.9 Manacher

```
1 S[1] = '%';
2 for(int i = 1; i <= len; i++) {
3     S[i << 1] = 'g';
4     S[i << 1|1] = s[i];
5 }
6 len = len << 1 | 1;
7 S[++len] = 'g';
8 S[++len] = 'f';
9 int mx = 0, id = 0, ans = 0;
10 for(int i = 1; i <= len; i++) {
11     if(mx > i) p[i] = min(p[id * 2 - i], mx - i);
12     else p[i] = 1;
13     while(S[i - p[i]] == S[i + p[i]] ++p[i];
14     if(i + p[i] > mx) id = i, mx = i + p[i];
15     ans = max(ans, p[i] - 1);
16 }
```


4.10 Lyndon/最小表示法

```

1 vector <int> duval(vector <int> S) {
2     int i = 0, j, k, s = S.size(); vector <int> ans;
3     while(i < s) {
4         j = i, k = i + 1;
5         while(j < s && k < s && S[j] <= S[k]) {
6             if(S[j] == S[k]) ++ j;
7             | else j = i; ++ k;
8             } while (i <= j) { ans.pb(i + k - j - 1); i += k - j; }
9         } return ans; // [ans[i] + 1, ans[i + 1]] is a lyndon word
10    }
11    vector <int> min_rep(vector <int> S) {
12        int k = 0, i = 0, j = 1, n = S.size();
13        while (k < n && i < n && j < n) {
14            | if (S[(i + k) % n] == S[(j + k) % n]) k ++;
15            | else {
16                | S[(i + k) % n] > S[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
17                | if (i == j) i ++; k = 0;
18            }
19            | i = min(i, j);
20            rotate(S.begin(), S.begin() + i, S.end()); return S;
21        }

```

4.11 Runs

```

1 // need lcp and lcs
2 bool cmp(int x, int y) {
3     int l = lcp(x, y);
4     if(x + l > n) return true;
5     if(y + l > n) return false;
6     return S[x + l] < S[y + l];
7 }
8 set <pi> ex;
9 void ins(int l, int r) {
10     int p = r - l;
11     int l1 = lcp(l, r);
12     int l2 = lcs(l - 1, r - 1);
13     int l = l - l2, R = r + l1 - 1;
14     if(R - l + 1 >= 2 * p) {
15         | auto iter = ex.lower_bound(pi(l, R));
16         | if(iter != ex.end() && *iter == pi(l, R)) return ;
17         | ex.emplace_hint(iter, pi(l, R));
18         | runs.pb((run){l, R, p});
19     }
20 }
21 void Run(int o) {
22     static int s[N];
23     int top = 0; s[++top] = n + 1;
24     for(int i = n; i; i--) {
25         | while(top > 1 && cmp(i, s[top]) == o) --top;
26         | ins(i, s[top]), s[++top] = i;
27     }
28 }

```

5 数据结构

5.1 区间加区间求和树状数组

```

1 // 后缀加, 前缀求和
2 struct BIT {
3     ll a[N], b[N];
4     void add(ll p, int v) {
5         | for(int i = p; i < N; i += i & -i)
6             | a[i] += v, b[i] += p * v;
7     }
8     ll qry(ll p) {
9         | ll res = 0;
10        for(int i = p; i & -i) res += (p + 1) * a[i] - b[i];
11        return res;
12    }
13    void add(int l, int r, int v) { add(l, v), add(r + 1, -v); }
14    ll qry(int l, int r) { return qry(r) - qry(l - 1); }
15    bit;

```

5.2 zkw 线段树

```

1 struct seg {
2     ll o[1 << 20]; int L;
3     void upt(int x) {
4         | o[x] = o[x << 1] + o[x << 1 | 1];
5     }
6     void init(int n, int *w) {
7         | L = 2 << std::lg(n + 1);
8         | for(int i = 1; i <= n; ++i) o[i + L] = w[i];
9         | for(int i = L; i >= 1; --i) upt(i);
10    }
11    void upt(int p, int v) {
12        | for(o[p + L] += v; p >= 1; upt(p));
13    }
14    ll qry(int l, int r) {
15        | l += L - 1, r += L + 1;
16        | ll ans = 0;
17        | for(; l ^ r ^ 1; l >= 1, r >= 1) {
18            | if((l & 1) == 0) ans += o[l ^ 1];
19            | if((r & 1) == 1) ans += o[r ^ 1];
20        }
21        return ans;
22    }
23    // if there is no I
24    ll qry2(int l, int r) {
25        | if(l == r) return o[l + L];
26        | ll le = o[l + L], ri = o[r + L];
27        | l += L, r += L;
28        | for(; l ^ r ^ 1; l >= 1, r >= 1) {
29            | if((l & 1) == 0) le = le + o[l ^ 1];
30            | if((r & 1) == 1) ri = o[r ^ 1] + ri;
31        }
32        return le + ri;
33    }

```

```
34 } sgt;
```

5.3 Link Cut Tree

```
1 int son[N][2], fa[N], rev[N];
2 int get(int x, int p = 1) { return son[fa[x]][p] == x; }
3 void update(int x) { }
4 int is_root(int x) { return !get(x) || get(x, 0); }
5 void rotate(int x) {
6     | int y = fa[x], z = fa[y], b = get(x);
7     | if(!is_root(y)) son[z][get(y)] = x;
8     | son[y][b] = son[x][!b], son[x][!b] = y;
9     | fa[son[y][b]] = y, fa[y] = x, fa[x] = z;
10    | update(y);
11 }
12 void put(int x) {
13     | if(x) rev[x] ^= 1, std::swap(son[x][0], son[x][1]);
14 }
15 void down(int x) {
16     | if(rev[x]) {
17         | | put(son[x][0]);
18         | | put(son[x][1]);
19         | | rev[x] = 0;
20     }
21 }
22 void pushdown(int x) {
23     | if(!is_root(x)) pushdown(fa[x]);
24     | down(x);
25 }
26 void splay(int x) {
27     | for(pushdown(x);!is_root(x);rotate(x)) if(!is_root(fa[x]))
28     | | rotate(get(x) ^ get(fa[x]) ? x : fa[x]);
29     | update(x);
30 }
31 void access(int x) {
32     | for(int t = 0; x; son[x][1] = t, t = x, x = fa[x])
33     | | splay(x);
34 }
35 void makeroot(int x) {
36     | access(x), splay(x), put(x);
37 }
```

5.4 FHQ Treap

```
1 int root, cc;
2 struct hh { int w, pri, ch[2], size; } tr[lsz];
3 #define ls(x) tr[x].ch[0]
4 #define rs(x) tr[x].ch[1]
5 void pushup(int x) { tr[x].size = 1 + tr[ls(x)].size + tr[rs(x)].size; }
6 int newnode(int w) {
7     | ++cc;
8     | tr[cc].w = w, tr[cc].pri = rnd(1, int(1e9)), tr[cc].size = 1;
9     | return cc;
10 }
11 int merge(int x, int y) {
```

```
12 | if (!x || !y) return x+y;
13 | if (tr[x].pri < tr[y].pri) return rs(x) = merge(rs(x), y), pushup(x), x;
14 | return ls(y) = merge(x, ls(y)), pushup(y), y;
15 }
```

```
16 void split(int x, int w, int &a, int &b) {
17     | if (!x) return a=b=0, void();
18     | if (tr[x].w <= w) a=x, split(rs(x), w, rs(x), b);
19     | else b=x, split(ls(x), w-a, ls(x));
20     | pushup(x);
21 }
```

5.5 pbds tree

```
1 #include <bits/extc++.h>
2 using namespace __gnu_pbds;
3 template<class T> // insert, erase, join, order_of_key, find_by_order (return
4     ↳ iterator), order is 0-index
5 using Tree = tree<T, null_type, std::less<T>, rb_tree_tag,
6     ↳ tree_order_statistics_node_update>;
```

6 geometry

6.1 向量

```
1 using db = long double;
2 const db eps = 1e-10;
3 db sgn(db x) { return x < -eps ? -1 : x > eps; }
4 db eq(db x, db y) { return !sgn(x - y); }
5 struct p2 {
6     | db x, y;
7     | db norm() const { return x * x + y * y; }
8     | db abs() const { return std::sqrt(x * x + y * y); }
9     | db arg() const { return atan2(y, x); }
10 };
11 db arg(p2 x, p2 y) {
12     | db a = y.arg() - x.arg();
13     | if(a > pi) a -= pi * 2;
14     | if(a < -pi) a += pi * 2;
15     | return a;
16 }
17 p2 r90(p2 x) { return {-x.y, x.x}; }
18 p2 operator + (p2 x, p2 y) { return {x.x + y.x, x.y + y.y}; }
19 p2 operator - (p2 x, p2 y) { return {x.x - y.x, x.y - y.y}; }
20 p2 operator / (p2 x, db y) { return {x.x / y, x.y / y}; }
21 p2 operator * (p2 x, db y) { return {x.x * y, x.y * y}; }
22 p2 operator * (db y, p2 x) { return {x.x * y, x.y * y}; }
23 db operator * (p2 x, p2 y) { return {x.x * y.y - x.y * y.x}; }
24 db operator % (p2 x, p2 y) { return x.x * y.x + x.y * y.y; }
25 int half(p2 x) { return x.y < 0 || (x.y == 0 && x.x <= 0); }
26 int half(p2 x) { return x.y < -eps || (std::fabs(x.y) < eps && x.x < eps); }
27 bool cmp(p2 a, p2 b) { return half(a) == half(b) ? a * b > 0 : half(b); }
28 bool cmp_eq(p2 A, p2 B) { return half(A) == half(B) && eq(A * B, 0); }
29 // 判断 A, B, C 三个向量是否是逆时针顺序
30 // 如果是, 返回 1
31 // 如果 (A, B), (C, B) 同方向共线, 返回 -1
```

```

32 // 如果是顺时针, 返回 0
33 bool cmp_ct(p2 A, p2 B, p2 C) {
34     if(cmp_eq(A, B)) return -1;
35     if(cmp_eq(C, B)) return -1;
36     if(cmp(A, B)) {
37         return cmp(B, C) || cmp(C, A);
38     } else {
39         return cmp(B, C) && cmp(C, A);
40     }
41 }
42 // 凸包 DP
43 struct pr { int i, j; p2 get() const { return a[j] - a[i]; } };
44 bool cmpseg(pr x, pr y) {
45     p2 A = x.get(), B = y.get();
46     if(!cmp(A, B) && !cmp(B, A)) return a[x.i] % A < a[y.i] % A;
47     return cmp(A, B);
48 }

```

6.2 直线半平面

```

1 struct line : p2 {
2     db z;
3     // a * x + b * y + c (= or >) 0
4     line() = default;
5     line(db a, db b, db c) : p2{a, b}, z(c) {}
6     line(p2 a, p2 b) : p2(r0(b - a)), z(a * b) {} //左侧 > 0
7     db operator()(p2 a) const { return a % p2(*this) + z; }
8     line perp() const { return {y, -x, 0}; } // 垂直
9     line para(p2 o) { return {x, y, z - (*this)(o)}; } // 平行
10 };
11 p2 operator & (line x, line y) {
12     return p2{p2{x.z, x.y} * p2{y.z, y.y}, p2{x.x, x.z} * p2{y.x, y.z}} /
13         -(p2{x} * p2{y});
14     // 注意此处精度误差较大, 以及 res.y 需要较高精度
15 }
16 p2 proj(p2 x, line l){return x - p2(l) * (l(x) / l.norm());} //投影
17 p2 refl(p2 x, line l){return x - p2(l) * (l(x) / l.norm()) * 2;} //对称
18 bool is_para(line x, line y){return eq(p2(x) * p2(y), 0);} //判断线平行
19 bool is_perp(line x, line y){return eq(p2(x) % p2(y), 0);} //判断线垂直
20 bool online(p2 x, line l) { return eq(l(x), 0); } // 判断点在线上
21 int ccw(p2 a, p2 b, p2 c) {
22     int sign = sgn((b - a) * (c - a));
23     if(sign == 0) {
24         if(sgn((b - a) % (c - a)) == -1) return 2;
25         if((c - a).norm() > (b - a).norm() + eps) return -2;
26     }
27     return sign;
28 }
29 db det(line a, line b, line c) {
30     p2 A = a, B = b, C = c;
31     return c.z * (A * B) + a.z * (B * C) + b.z * (C * A);
32 }
33 db check(line a, line b, line c) { // sgn same as c(a & b), 0 if error

```

```

34 | return sgn(det(a, b, c)) * sgn(p2(a) * p2(b));
35 }
36 bool paras(line a, line b) { // 射线同向
37     return is_para(a, b) && p2(a) % p2(b) > 0;
38 }

```

6.3 半平面交

```

1 std::vector<p2> HPI(std::vector<line> vs) {
2     auto cmp = [](line a, line b) {
3         if(paras(a, b)) return dist(a) < dist(b);
4         return ::cmp(p2(a), p2(b));
5     };
6     sort(vs.begin(), vs.end(), cmp);
7     int ah = 0, at = 0, n = size(vs);
8     std::vector<line> deq(n + 1);
9     std::vector<p2> ans(n);
10    deq[0] = vs[0];
11    for(int i = 1; i <= n; ++i) {
12        line o = i < n ? vs[i] : deq[ah];
13        if(paras(vs[i - 1], o)) continue;
14        for(ah < at && check(deq[at - 1], deq[at], o) < 0;) -- at; //maybe <=
15        if(i == n) for(ah < at && check(deq[at], deq[at + 1], o) < 0;) ++ ah;
16        if(!is_para(o, deq[at])) {
17            ans[at] = o & deq[at];
18            | deq[++at] = o;
19        }
20    }
21    if(at - ah <= 2) return {};
22    return {ans.begin() + ah, ans.begin() + at};
23 }

```

6.4 线段

```

1 struct seg {
2     p2 x, y;
3     seg() {}
4     seg(const p2 & A, const p2 & B) : x(A), y(B) {}
5     bool onseg(const p2 & o) const {
6         return (o - x) % (o - y) < eps && std::fabs((o - x) * (o - y)) < eps;
7     }
8 };
9 db dist(const seg & o, const p2 & x) {
10     if((o.x - o.y) % (x - o.y) <= eps) return (x - o.y).abs();
11     if((o.y - o.x) % (x - o.x) <= eps) return (x - o.x).abs();
12     return fabs((o.x - x) * (o.y - x) / (o.x - o.y).abs());
13 }
14 bool is_isc(const seg & x, const seg & y) {
15     return
16     | ccw(x.x, x.y, y.x) * ccw(x.x, x.y, y.y) <= 0 &&
17     | ccw(y.x, y.y, x.x) * ccw(y.x, y.y, x.y) <= 0;
18 }
19 db dist(const seg & x, const seg & y) {
20     if(is_isc(x, y)) return 0;
21     return std::min({dist(y, x.x), dist(y, x.y), dist(x, y.x), dist(x, y.y)});

```

```
22 }
}
```

6.5 多边形

```
1 using polygon = std::vector<p2>;
2 // counter-clockwise
3 db area(const polygon & x) {
4     db res = 0;
5     for(int i = 2; i < (int) x.size(); ++i) {
6         res += (x[i - 1] - x[0]) * (x[i] - x[0]);
7     }
8     return res / 2;
9 }
10 bool is_convex(const polygon & x, bool strict = 1) {
11     // warning, maybe wrong
12     const db z = strict ? eps : -eps;
13     for(int i = 2; i < (int) x.size() + 2; ++i) {
14         if((x[(i - 1) % x.size()] - x[i - 2]) * (x[i % x.size()] - x[i - 2]) <
15             ↪ z) return 0;
16     }
17     return 1;
18 }
19 int contain(const std::vector<p2> & a, p2 o) { // 简单多边形包含判定
20     bool in = 0;
21     for(int i = 0; i < (int) a.size(); ++i) {
22         p2 x = a[i] - o, y = a[(i + 1) % a.size()] - o;
23         if(x.y > y.y) std::swap(x, y);
24         if(x.y <= eps && y.y > eps && x * y < -eps) in ^= 1;
25         if(std::fabs(x * y) < eps && x % y < eps) return 2; // 在线段上, 看情况改
26     }
27     return in;
28 }
```

6.6 线段 in 多边形

```
1 bool contains(p2 x, p2 y, const std::vector<p2> & a) {
2     using pr = std::pair<double, int>;
3     std::vector<pr> e = {pr(-inf, 0), pr(inf, 0)};
4     p2 t = y - x;
5     auto f = [6](p2 a, p2 b, p2 c, p2 d) {
6         return (b - a).abs() * ((c - a) * (d - c)) / ((b - a) * (d - c));
7     };
8     for(int i = 0; i < (int) a.size(); ++i) {
9         p2 u = a[i], v = a[(i + 1) % a.size()];
10        int a = sgn(t * (u - x));
11        int b = sgn(t * (v - x));
12        if(a != b) e.emplace_back(f(x, y, u, v), b - a);
13    }
14    sort(e.begin(), e.end());
15    int sum = 0; db R = t.abs();
16    for(int i = 0; i + 1 < (int) e.size(); ++i) {
17        sum += e[i].second;
18        if(sum == 0 && std::max(e[i].first, 0.) + eps < std::min(e[i +
19            ↪ 1].first, R)) {
20            return 0;
21        }
22    }
```

```
20 | | | }
21 | | }
22 | return 1;
23 }
```

6.7 图形求交

```
1 struct circle : p2 { db r; };
2 std::vector<p2> operator & (circle o, line l) {
3     p2 v = l, Rv = r*o(v); db L = l.abs();
4     db d = l(p2(o)) / L, x = o.r * o.r - d * d;
5     if(x < -eps) return {};
6     x = std::sqrt(x * sgn(x));
7     p2 z = p2(o) - v * (d / L), p = Rv * (x / L);
8     return {z + p, z - p};
9     // 1 如果是构造函数给出, 那么返回交点按射线顺序
10    std::vector<p2> operator & (circle o, seg s) {
11        std::vector<p2> b;
12        for(p2 x : (o & s.to_l()))
13            if(s.onseg(x)) b.push_back(x);
14        return b;
15    }
16    std::vector<p2> operator & (circle o0, circle o1) {
17        p2 tmp = (p2(o1) - p2(o0)) * 2.;
18        return o0 & line(tmp.x, tmp.y, o1.r * o1.r - o0.r * o0.r + o0.norm() -
19            ↪ o1.norm());
20    }
21    std::vector<p2> tang(circle o, p2 x) {
22        db d = (x - p2(o)).abs();
23        if(d <= o.r + eps) return {};
24        return o & circle{x, sqrt(d * d - o.r * o.r)};
25    }
26    // 三角形 (0, a, b) 和圆 o 的交的有向面积 * 2
27    db intersect(circle o, p2 a, p2 b) {
28        a = a - p2(o), b = b - p2(o); o.x = o.y = 0;
29        int va = a.abs() <= o.r + eps;
30        int vb = b.abs() <= o.r + eps;
31        if(va && vb) return a * b;
32        auto v = o & seg{a, b}; // 注意这里, 有必要改一下 onseg, 去掉平行判定
33        if(v.empty()) return arg(a, b) * o.r * o.r;
34        db sum = 0;
35        sum += va ? a * v[0] : arg(a, v[0]) * o.r * o.r;
36        sum += vb ? v.back() * b : arg(v.back(), b) * o.r * o.r;
37        if(v.size() > 1) sum += v[0] * v[1];
38        return sum;
39    }
40    // 有向弓形面积 * 2, arg 不能改
41    db csegS(circle o, p2 a, p2 b) {
42        a = a - p2(o);
43        b = b - p2(o);
44        db d = b.arg() - a.arg();
45        if(d < 0) d += pi * 2;
46        return d * o.r * o.r - a * b;
47    }
```

19

```
47 // 两圆交的面积 * 2
48 db intersect(circle o0, circle o1) {
49     if(o0.r > o1.r) std::swap(o0, o1);
50     db d = (p2(o0) - p2(o1)).abs();
51     if(d <= (o1.r - o0.r) + eps) return 2 * pi * o0.r * o0.r;
52     if(d >= o1.r + o0.r - eps) return 0;
53     auto v = o0 & o1;
54     return csegs(o0, v[1], v[0]) + csegs(o1, v[0], v[1]);
55 }
```

6.8 凸包

结果为逆时针。

```
1 db cross(p2 x, p2 y, p2 z) { return (y.x - x.x) * (z.y - x.y) - (y.y - x.y) *
    ↪ (z.x - x.x); }
2 std::vector<p2> gethull(std::vector<p2> o) {
3     rgs::sort(o, [(p2 x, p2 y) { return eq(x.x, y.x) ? x.y < y.y : x.x < y.x;
    ↪ }]);
4     o.erase(unique(o.begin(), o.end(), [(p2 x, p2 y) {
5         return eq(x.x, y.x) && eq(x.y, y.y);
6     }]), o.end());
7     std::vector<p2> s;
8     for(int i = 0; i < (int) o.size(); ++i) {
9         for(; s.size() >= 2 && cross(s.rbegin()[1], s.back(), o[i]) <= eps; )
10             s.pop_back();
11         s.push_back(o[i]);
12     }
13     for(int i = o.size() - 2, t = s.size(); i >= 0; --i) {
14         for(; s.size() > t && cross(s.rbegin()[1], s.back(), o[i]) <= eps; )
15             s.pop_back();
16         s.push_back(o[i]);
17     }
18     if(s.size() > 1) s.pop_back();
19     return s;
20 } // 把两个 eps 改成 -eps 可求出所有在凸包上的点
21 int findmin(std::vector<p2> &a, auto cmp) {
22     int l = 0, r = a.size() - 1, d = 1;
23     if(cmp(a.back(), a[0])) std::swap(l, r), d = -1;
24     for(; (r - l) * d > 1; ) {
25         int mid = (l + r) >> 1;
26         if(cmp(a[mid], a[mid - d]) && cmp(a[mid], a[l])) {
27             l = mid;
28         } else {
29             r = mid;
30         }
31     }
32     return l;
33 } // cmp is less, and a.size()>0 plz
34 int contains(std::vector<p2> &a, p2 x) {
35     auto it = lower_bound(a.begin() + 2, a.end(), x, [&](p2 x, p2 y) {
36         return cross[a[0], x, y] > 0;
37     });
38     if(*it < x) return 0;
39     if(*it > x) return 1;
40     if(*it == x) return 2;
41     return 0;
42 }
```

```
39 | if(it != a.end()) && c0 >= 0 && c1 >= 0) {
40 |     return c0 > 0 && c1 > 0 && cross(a.back(), a[0], x) > 0 ? IN : ON;
41 | } else {
42 |     return 0;
43 | }
44 | // a.size()>2 plz
```

6.9 上凸壳

结果显然为顺时针。

```
1 std::vector<p2> gethull(std::vector<p2> o) {
2     sort(o.begin(), o.end(), [(p2 x, p2 y) {
3         if(x.x == y.x) {
4             return x.y > y.y; // gt => lt
5         } else {
6             return x.x < y.x;
7         }
8     }]);
9     std::vector<p2> stack;
10    for(p2 x : o) {
11        if(stack.size() && stack.back().x == x.x) {
12            continue;
13        }
14        for(; stack.size() >= 2 && cross(stack.rbegin()[1], stack.back(), x) >=
    ↪ 0; ) { // gt => lt
15            stack.pop_back();
16        }
17        stack.push_back(x);
18    }
19    return stack;
20 }
```

6.10 最小圆覆盖

```
1 struct circle : p2 { db r; };
2 circle incircle(p2 a, p2 b, p2 c) {
3     db A = (b - c).abs(), B = (c - a).abs(), C = (a - b).abs();
4     return {(a * A + b * B + c * C) / (A + B + C), fabs((b - a) * (c - a)) /
    ↪ (A + B + C)};
5 } // 三点确定内心, 不是最小圆覆盖内容
6 circle circumcenter(p2 a, p2 b, p2 c) {
7     p2 bc = c - b, ca = a - c, ab = b - a;
8     p2 o = (b + c - r90(bc)) * (ca % ab) / (ca * ab) / 2;
9     return {o, (a - o).abs()};
10 } // 三点确定外心
11 circle cir(p2 a, p2 b) { // 根据直径生成圆
12     return {(a + b) / 2, (a - b).abs() / 2};
13 }
14 bool in(circle x, p2 y) { return (p2(x) - y).abs() <= x.r + eps; }
15 circle mincircle(std::vector<p2> a) { // 最小圆覆盖, 需要 shuffle
16     circle o = {a[0], 0};
17     int n = a.size();
18     for(int i = 1; i < n; ++i) {
19         if(in(o, a[i])) continue;
20         o = cir(a[0], a[i]);
21     }
```

```
21 |         for(int j = 1;j < i;++j) {
22 |             | if(in(o, a[j])) continue;
23 |             | o = cir(a[j], a[i]);
24 |             | for(int k = 0;k < j;++k) {
25 |                 | if(in(o, a[k])) continue;
26 |                 | o = circumcenter(a[i], a[j], a[k]);
27 |             | }
28 |         }
29 |     }
30 |     return o;
31 | }
```

6.11 最近点对

```
1 | db mindist(std::vector<p2> a) {
2 |     db ans = 1e18;
3 |     sort(a.begin(), a.end(), [](p2 x, p2 y) { return x.x < y.x; });
4 |     ans = (a[0] - a[1]).abs();
5 |     auto solve = [&](auto s, int l, int r) {
6 |         | if(l + 1 == r) return ;
7 |         | int mid = (l + r) >> 1;
8 |         | db mx = a[mid].x;
9 |         | s(s, l, mid), s(s, mid, r);
10 |        | static std::vector<p2> b; b.clear();
11 |        | inplace_merge(a.begin() + l, a.begin() + mid, a.begin() + r, [](p2 x,
12 |            | ↪ p2 y) { return x.y < y.y; });
13 |        | for(int i = l;i < r;++i)
14 |            | | if(fabs(a[i].x - mx) <= ans) b.push_back(a[i]);
15 |            | for(int i = 1;i < (int) b.size();++i)
16 |                | | for(int j = i - 1;j >= 0 && b[i].y <= b[j].y + ans;--j) ans =
17 |                    | ↪ std::min(ans, (b[i] - b[j]).abs());
18 |        | };
19 |        | solve(solve, 0, a.size());
20 |        | return ans;
21 |    }
```

6.12 凸包直径

```
1 | db convex_diameter(std::vector<p2> & o) {
2 |     int n = size(o);
3 |     db max = 0;
4 |     for(int i = 0, j = 0;i < n;++i) {
5 |         | for(;j + 1 < n && (o[j] - o[i]).abs() < (o[j + 1] - o[i]).abs();) ++ j;
6 |         | max = std::max(max, (o[j] - o[i]).abs());
7 |     }
8 |     return max;
9 | } // 凸包直径
```

6.13 切凸包

```
1 | std::vector<p2> cut(const std::vector<p2> & o, line l) {
2 |     std::vector<p2> res;
3 |     int n = size(o);
4 |     for(int i = 0;i < n;++i) {
5 |         | p2 a = o[i], b = o[(i + 1) % n];
6 |         | if(sgn(l(a)) >= 0) res.push_back(a); // 注意 sgn 精度
```

```
7 |         | if(sgn(l(a)) * sgn(l(b)) < 0) res.push_back(line(a, b) & l);
8 |         | }
9 |         | if(res.size() <= 2) return {};
10 |        | return res;
11 |    } // 切凸包
```

6.14 V 图

```
1 | std::vector<line> cut(const std::vector<line> & o, line l) {
2 |     std::vector<line> res;
3 |     int n = size(o);
4 |     for(int i = 0;i < n;++i) {
5 |         | line a = o[i], b = o[(i + 1) % n], c = o[(i + 2) % n];
6 |         | int va = check(a, b, l), vb = check(b, c, l);
7 |         | if(va > 0 || vb > 0 || (va == 0 && vb == 0)) {
8 |             | res.push_back(b);
9 |         }
10 |        | if(va >= 0 && vb < 0) {
11 |            | res.push_back(l);
12 |        }
13 |        | }
14 |        | if(res.size() <= 2) return {};
15 |        | return res;
16 |    } // 切凸包
17 |    line bisector(p2 a, p2 b) { return line(a.x - b.x, a.y - b.y, (b.norm() -
18 |        | ↪ a.norm()) / 2); }
```

```
1 | std::vector<std::vector<line>> voronoi(std::vector<p2> p) {
2 |     int n = p.size();
3 |     auto b = p; shuffle(b.begin(), b.end(), gen);
4 |     const db V = 1e5; // 边框大小, 重要
5 |     std::vector<std::vector<line>> a(n, {
6 |         | {V, 0, V * V}, {0, V, V * V},
7 |         | {-V, 0, V * V}, {0, -V, V * V},
8 |         | });
9 |     for(int i = 0;i < n;++i) {
10 |        | for(p2 x : b) if((x - p[i]).abs() > eps) {
11 |            | | a[i] = cut(a[i], bisector(p[i], x));
12 |        }
13 |    }
14 |    return a;
15 | }
```

6.15 Delaunay 三角剖分

```
1 | using i128 = __int128;
2 | using Q = struct Quad*;
3 | p2 arb(LLONG_MAX, LLONG_MAX);
4 | struct Quad {
5 |     | Q rot, o; p2 p = arb; bool mark;
6 |     | p2& F() { return r() -> p; }
7 |     | Q& r() { return rot->rot; }
8 |     | Q prev() { return rot->o->rot; }
9 |     | Q next() { return r()->prev(); }
10 |    } *H;
11 | ll cross(p2 a, p2 b, p2 c) {
```

```

12 | return (b - a) * (c - a);
13 | }
14 | bool circ(p2 p, p2 a, p2 b, p2 c) { // p 是否在 a, b, c 外接圆中
15 |     i128 p2 = p.norm(), A = a.norm() - p2, B = b.norm() - p2, C = c.norm() -
        |     ↪ p2;
16 |     a = a - p, b = b - p, c = c - p;
17 |     return (a * b) * C + (b * c) * A + (c * a) * B > 0;
18 | }
19 | Q link(p2 orig, p2 dest) {
20 |     Q r = H ? H : new Quad{new Quad{new Quad{0}}}};
21 |     H = r -> o; r -> r() -> r() = r;
22 |     for(int i = 0; i < 4; ++i)
23 |         | r = r -> rot, r -> p = arb, r -> o = i & 1 ? r : r -> r();
24 |     r -> p = orig, r -> F() = dest;
25 |     return r;
26 | }
27 | void splice(Q a, Q b) {
28 |     std::swap(a -> o -> rot -> o, b -> o -> rot -> o);
29 |     std::swap(a -> o, b -> o);
30 | }
31 | Q conn(Q a, Q b) {
32 |     Q q = link(a -> F()), b -> p);
33 |     splice(q, a -> next());
34 |     splice(q -> r(), b);
35 |     return q;
36 | }
37 | std::pair<Q, Q> rec(const std::vector<p2> & s) {
38 |     int N = size(s);
39 |     if(N <= 3) {
40 |         | Q a = link(s[0], s[1]), b = link(s[1], s.back());
41 |         | if(N == 2) return {a, a -> r()};
42 |         | splice(a -> r(), b);
43 |         | ll side = cross(s[0], s[1], s[2]);
44 |         | Q c = side ? conn(b, a) : 0;
45 |         | return {side < 0 ? c -> r() : a, side < 0 ? c : b -> r()};
46 |     }
47 |     #define H(e) e -> F(), e -> p
48 |     #define valid(e) (cross(e -> F(), H(base)) > 0)
49 |     int half = N / 2;
50 |     auto [ra, A] = rec({s.begin(), s.end() - half});
51 |     auto [B, rb] = rec({s.end() - half, s.end()});
52 |     while((cross(B -> p, H(A)) < 0 && (A = A -> next())) ||
53 |         | (cross(A -> p, H(B)) > 0 && (B = B -> r() -> o)));
54 |     Q base = conn(B -> r(), A);
55 |     if(A -> p == ra -> p) ra = base -> r();
56 |     if(B -> p == rb -> p) rb = base;
57 |     #define DEL(e, init, dir) Q e = init -> dir; if(valid(e)) \
58 |         | for(circ(e -> dir -> F()), H(base), e -> F()); { \
59 |         |     Q t = e -> dir; \
60 |         |     splice(e, e -> prev()); \
61 |         |     splice(e -> r(), e -> r() -> prev()); \
62 |         |     e -> o = H, H = e, e = t; \
63 |     }

```

```

64 | for(;;) {
65 |     | DEL(LC, base -> r(), o);
66 |     | DEL(RC, base, prev());
67 |     | if(!valid(LC) && !valid(RC)) break;
68 |     | if(!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
69 |         | base = conn(RC, base -> r());
70 |     | else
71 |         | base = conn(base -> r(), LC -> r());
72 |     }
73 |     return {ra, rb};
74 | }
75 | std::vector<p2> triangulate(std::vector<p2> a) {
76 |     sort(a.begin(), a.end()); // unique
77 |     if((int)size(a) < 2) return {};
78 |     Q e = rec(a).first;
79 |     std::vector<Q> q = {e};
80 |     while(cross(e -> o -> F()), e -> F(), e -> p) < 0) e = e -> o;
81 |     #define ADD { Q c = e; do { c -> mark = 1; a.push_back(c -> p); \
82 |         | q.push_back(c -> r());, c = c -> next(); } while(c != e); }
83 |     ADD; a.clear();
84 |     for(int qi = 0; qi < (int) size(q);) if(! (e = q[qi++]) -> mark) ADD;
85 |     return a;
86 | } // 返回若干逆时针三角形 {t[0][0], t[0][1], t[0][2], t[1][0], \dots\}

```

7 geometry3d

7.1 向量

```

1 | struct p3 {
2 |     | db x, y, z;
3 |     | db norm() const { return x * x + y * y + z * z; }
4 |     | db abs() const { return std::sqrt(norm()); }
5 | };
6 | p3 operator + (p3 x, p3 y){ return {x.x + y.x, x.y + y.y, x.z + y.z}; }
7 | p3 operator - (p3 x, p3 y){ return {x.x - y.x, x.y - y.y, x.z - y.z}; }
8 | p3 operator * (p3 x, db y) { return {x.x * y, x.y * y, x.z * y}; }
9 | p3 operator / (p3 x, db y) { return {x.x / y, x.y / y, x.z / y}; }
10 | p3 operator * (p3 x, p3 y) { // 三维叉积需要更高的精度
11 |     | return {
12 |         | x.y * y.z - x.z * y.y,
13 |         | x.z * y.x - x.x * y.z,
14 |         | x.x * y.y - x.y * y.x
15 |     };
16 | }
17 | db operator % (p3 x, p3 y) { return x.x * y.x + x.y * y.y + x.z * y.z; }
18 | p3 pervec(p3 x) {
19 |     | return fabs(x.x) > fabs(x.z) ? p3{ x.y, -x.x, 0 } : p3{0, -x.z, x.y};
20 | } // 找到一个与给定向量垂直的向量
21 | db area(p3 a, p3 b, p3 c) { return ((b - a) * (c - a)).abs(); } // 三角形面积两
    |     ↪ 倍
22 | db volume(p3 d, p3 a, p3 b, p3 c) { // 四面体有向体积六倍
23 |     | return (d - a) % ((b - a) * (c - a));
24 | }

```


7.2 平面

```

1 struct plane {
2     p3 n; db d; // n dot x = d
3     plane() {}
4     plane(p3 a, p3 b, p3 c) : n((c - a) * (b - a)) { d = n % a; }
5     db side(p3 x) const { return n % x - d; }
6     db dist(p3 w) const { return side(w) / n.abs(); }
7     p3 proj(p3 w) const { return w - n * (side(w) / n.abs()); }
8 };

```

7.3 直线

```

1 struct line3 {
2     p3 d, o; // kd + o
3     line3() {}
4     line3(p3 p, p3 q) : d(q - p), o(p) {}
5     line3(plane p1, plane p2) : d(p1.n * p2.n) { // 平面交出直线
6         | o = (p2.n * p1.d - p1.n * p2.d) * d / d.norm();
7     }
8     db dist(p3 p) const { return (d * (p - o)).abs() / d.abs(); }
9     p3 proj(p3 p) const { return o + d * (d % (p - o)) / d.norm(); } // 投影
10    p3 relf(p3 p) const { return proj(p) * 2 - p; } // 对称
11    p3 operator & (const plane & p) const { // 线与平面交
12        | return o - d * p.side(o) / (p.n % d);
13    }
14 };
15 db dist(line3 l1, line3 l2) {
16     p3 n = l1.d * l2.d;
17     if(n.abs() < eps) return l1.dist(l2.o);
18     return abs((l2.o - l1.o) % n) / n.abs();
19 }
20 p3 closestOnL1(line3 l1, line3 l2) {
21     p3 n2 = l2.d * (l1.d * l2.d);
22     return l1.o + l1.d * ((l2.o - l1.o) % n2) / (l1.d % n2);
23 }
24 bool ispara(plane p1, plane p2){return(p1.n * p2.n).abs() < eps;} //判断是否平行
25 bool ispara(line3 p1, line3 p2){return(p1.d * p2.d).abs() < eps;} //判断是否平行
26 bool isperp(plane p1, plane p2){return fabs(p1.n % p2.n) < eps;} //判断是否垂直
27 bool isperp(line3 p1, line3 p2){return fabs(p1.d % p2.d) < eps;} //判断是否垂直
28 line3 perthrough(plane p, p3 o){return line3(o, o + p.n);} //过平面一点做垂线

```

7.4 凸包

```

1 const int N = 2005;
2 struct face { int a[3]; plane p; };
3 int vis[N][N];
4 std::vector<face> f;
5 void convex3d(const std::vector<p3> & a) { // need to deal coplane
6     if(a.size() < 3) return;
7     auto getface = [&](int i, int j, int k) -> face { return {i, j, k,
8         | f = {getface(0, 1, 2), getface(0, 2, 1)};
9         | std::vector<face> tmp[2];
10        for(int i = 3; i < (int) a.size(); ++i) {
11            | for(auto x : f) {

```

```

12         | if(x.p.dist(a[i]) < -eps) {
13             | tmp -> push_back(x);
14         | } else {
15             | tmp[i].push_back(x);
16         | for(int t : {0, 1, 2}) vis[x.a[t]][x.a[(t + 1) % 3]] = i;
17         | }
18     }
19     for(auto x : tmp[1]) {
20         | for(int t : {0, 1, 2}) {
21             | if(vis[x.a[t]][x.a[(t + 1) % 3]] == i && vis[x.a[(t + 1) % 3]]
22                 | ->[x.a[t]] != i)
23                 | tmp[0].push_back(getface(x.a[t], x.a[(t + 1) % 3], i));
24         | }
25         | f = tmp[0]; tmp[0].clear(); tmp[1].clear();
26     }
27     for(int i = 0; i < (int)a.size(); ++i) memset(vis[i], 0, a.size() << 2);
28 }

```

8 Misc

8.1 Pragma

```

1 #pragma GCC optimize("Ofast")
2 #pragma GCC optimize("unroll-loops")
3 #pragma GCC target("sse, sse2, sse3, sse3.5, sse4, popcnt, abm, mmx, avx, avx2")
4 #pragma pack(1) // default=8

```

8.2 Barrett

```

1 struct DIV {
2     u64 x;
3     void init(u64 v) { x = -1ull / v; }
4     }; // 带误差版本 x = -1ull/v;
5     // ret=ans while x*(y-1)<2^64, ans-1<ret<=ans while x<2^64
6     u64 operator / (const u64 & x, const DIV & y) {
7         | return (u128) x * y.x >> 64;
8     }

```

8.3 LCS

```

1 int ltm;
2 struct bitset {
3     static const int B = 63;
4     u64 a[N / B + 1];
5     void set(int p) { a[p / B] |= 1ull << (p % B); }
6     bool test(int p) { return a[p / B] >> (p % B) & 1; }
7     void run(const bitset & o) {
8         | u64 c = 1;
9         | for(int i = 0; i < ltm; ++i) {
10             | u64 x = a[i], y = x | o.a[i];
11             | x += x + c + (~y & (1ull << 63)) - 1;
12             | a[i] = x & y, c = x >> 63;
13         | }
14     }
15 } dp;

```


8.4 日期公式

```
1 // Mon = 0, ... % 7
2 // days since 1/1/1
3 int getday(int y, int m, int d) {
4     if(m < 3) --y, m += 12;
5     return (365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d
6         - 307);
7 }
8 void date(int n, int &y, int &m, int &d) {
9     n += 429 + ((4 * n + 1227) / 146097 + 1) * 3 / 4;
10    y = (4 * n - 489) / 1461;
11    n -= y * 1461 / 4;
12    m = (5 * n - 1) / 153;
13    d = n - m * 153 / 5;
14    if (--m > 12) m -= 12, ++y;
15 }
```

8.5 Xorshift

```
1 u64 xorshift(u64 x) { x ^= x << 13; x ^= x >> 7; x ^= x << 17; return x; }
2 u32 xorshift(u32 x) { x ^= x << 13; x ^= x >> 17; x ^= x << 5; return x; }
```

9 配置

9.1 vimrc

```
1 set si ci ts=4 sw=4 nu cino=j1 backup undofile
2 syntax on
3 map<F9> <ESC>:!make %<<CR>
4 map<F10> <ESC>:!.%<<CR>
5 map<F4> <ESC>:!gdb %<<CR>
```

9.2 bashrc

```
1 export CXXFLAGS='-g -Wall -fsanitize=address,undefined -Dzqj -std=gnu++20'
2 mk() { g++ -O2 -Dzqj -std=gnu++20 $1.cpp -o $1; }
3 ulimit -s 1048576
4 ulimit -v 1048576
```

9.3 对拍

需要 chmod +x

```
1 while true; do
2     | ./gen > 1.in
3     | ./naive < 1.in > std.out
4     | ./a < 1.in > 1.out
5     | if diff 1.out std.out; then
6         | echo ac
7     | else
8         | echo wa
9         | break
10    | fi
11 done
```

9.4 编译参数

-D_GLIBCXX_DEBUG : STL debug mode
-fsanitize=address : 内存错误检查
-fsanitize=undefined : UB 检查

9.5 随机素数

979345007 986854057502126921
935359631 949054338673679153
931936021 989518940305146613
984974633 972090414870546877
984858209 956380060632801307

9.6 常数表

n	$\log_{10} n$	$n!$	$C(n, n/2)$	$\text{LCM}(1 \dots n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10		3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.44e14	5604
P_n	3733840	20422650	96646760	190569292100	1e9114
$n \leq$	10	100	1e3	1e4	1e5
$\max \omega(n)$	2	3	4	5	6
$\max d(n)$	4	12	32	64	128
$\pi(n)$	4	25	168	1229	9592
$n \leq$	1e7	1e8	1e9	1e10	1e11
$\max \omega(n)$	8	8	9	10	11
$\max d(n)$	448	768	1344	2304	4032
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9
$n \leq$	1e13	1e14	1e15	1e16	1e17
$\max \omega(n)$	12	12	13	13	14
$\max d(n)$	10752	17280	26880	41472	64512
$\pi(n)$					103680

10 注意事项

10.1 测试项目

phds tree, float128, int128, long double submit 命令, printf, MLE ?= RE, pragma, axv2, python,

10.2 bugs

看数据范围 (多测总和), 变量 shadow, 清空, long long, 数组大小, 模数, MLE?, 对拍记得看输出在不在变, 输出格式, inf 开小, 答案初值, STL 重构导致引用失效, 极端情况 (n=1)

11 tables

11.1 导数积分

$(\frac{u}{v})' = \frac{u'v-u v' }{v^2}$	$(\arctan x)' = \frac{1}{1+x^2}$	$(\operatorname{arcsinh} x)' = \frac{1}{\sqrt{1+x^2}}$
$(a^x)' = (\ln a)a^x$	$(\operatorname{arccot} x)' = -\frac{1}{1+x^2}$	$(\operatorname{arccosh} x)' = \frac{1}{\sqrt{x^2-1}}$
$(\tan x)' = \sec^2 x$	$(\operatorname{arccsc} x)' = -\frac{1}{1+x^2}$	$(\operatorname{arctanh} x)' = \frac{1}{1-x^2}$
$(\cot x)' = \csc^2 x$	$(\sec x)' = \tan x \sec x$	$(\operatorname{arccoth} x)' = \frac{1}{1-x^2}$
$(\sec x)' = \tan x \sec x$	$(\csc x)' = -\cot x \csc x$	$(\operatorname{arcsch} x)' = -\frac{1}{x\sqrt{1-x^2}}$
$(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$	$(\tanh x)' = \operatorname{sech}^2 x$	$(\operatorname{arcsch} x)' = -\frac{1}{x\sqrt{1-x^2}}$
$(\arccos x)' = -\frac{1}{\sqrt{1-x^2}}$	$(\coth x)' = -\operatorname{csch}^2 x$	$(\operatorname{sech} x)' = -\operatorname{sech} x \tanh x$
	$(\operatorname{csch} x)' = -\operatorname{csch} x \coth x$	$(\operatorname{arcsch} x)' = -\frac{1}{x\sqrt{1-x^2}}$

$ax^2+bx+c(a>0)$

$$\begin{aligned} 1. \int \frac{dx}{ax^2+bx+c} &= \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2 < 4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln \left| \frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}} \right| + C & (b^2 > 4ac) \end{cases} \\ 2. \int \frac{x}{ax^2+bx+c} dx &= \frac{1}{2a} \ln |ax^2+bx+c| - \frac{b}{2a} \int \frac{dx}{ax^2+bx+c} \end{aligned}$$

$\sqrt{\pm ax^2+bx+c(a>0)}$

$$\begin{aligned} 1. \int \frac{dx}{\sqrt{ax^2+bx+c}} &= \frac{1}{\sqrt{a}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C \\ 2. \int \sqrt{ax^2+bx+c} dx &= \frac{2ax+b}{4a} \sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a}^3} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C \\ 3. \int \frac{x}{\sqrt{ax^2+bx+c}} dx &= \frac{1}{a} \sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a}^3} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C \\ 4. \int \frac{dx}{\sqrt{c+bx-ax^2}} &= -\frac{1}{\sqrt{a}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C \\ 5. \int \sqrt{c+bx-ax^2} dx &= \frac{2ax-b}{4a} \sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a}^3} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C \\ 6. \int \frac{x}{\sqrt{c+bx-ax^2}} dx &= -\frac{1}{a} \sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a}^3} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C \end{aligned}$$

$\sqrt{\pm \frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$

$$\begin{aligned} 1. \int \frac{dx}{\sqrt{(x-a)(b-x)}} &= 2 \arcsin \sqrt{\frac{x-a}{b-x}} + C \quad (a < b) \\ 2. \int \sqrt{(x-a)(b-x)} dx &= \frac{2x-a-b}{4} \sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{8\sqrt{a}^3} \arcsin \sqrt{\frac{x-a}{b-x}} + C \end{aligned}$$

三角函数的积分

$$\begin{aligned} 1. \int \tan x dx &= -\ln |\cos x| + C \\ 2. \int \cot x dx &= \ln |\sin x| + C \\ 3. \int \sec x dx &= \ln \left| \tan \left(\frac{x}{2} + \frac{\pi}{2} \right) \right| + C = \ln |\sec x + \tan x| + C \\ 4. \int \csc x dx &= \ln \left| \tan \frac{x}{2} \right| + C = \ln |\csc x - \cot x| + C \\ 5. \int \sec^2 x dx &= \tan x + C \end{aligned}$$

$$\begin{aligned} 6. \int \csc^2 x dx &= -\cot x + C \\ 7. \int \sec x \tan x dx &= \sec x + C \\ 8. \int \csc x \cot x dx &= -\csc x + C \\ 9. \int \sin^2 x dx &= \frac{x}{2} - \frac{1}{4} \sin 2x + C \\ 10. \int \cos^2 x dx &= \frac{x}{2} + \frac{1}{4} \sin 2x + C \\ 11. \int \sin^n x dx &= -\frac{1}{n} \sin^{n-1} x \cos x + \frac{n-1}{n} \int \sin^{n-2} x dx \\ 12. \int \cos^n x dx &= \frac{1}{n} \cos^{n-1} x \sin x + \frac{n-1}{n} \int \cos^{n-2} x dx \\ 13. \int \frac{dx}{\sin^n x} &= -\frac{1}{n-1} \frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\sin^{n-2} x} \\ 14. \int \frac{dx}{\cos^n x} &= \frac{1}{n-1} \frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\cos^{n-2} x} \\ 15. \int \cos^m x \sin^n x dx &= \begin{cases} \frac{1}{m+n} \cos^{m-1} x \sin^{n+1} x + \frac{m-1}{m+n} \int \cos^{m-2} x \sin^n x dx & m \text{ is odd} \\ -\frac{1}{m+n} \cos^{m+1} x \sin^{n-1} x + \frac{n-1}{m+1} \int \cos^m x \sin^{n-2} x dx & n \text{ is odd} \end{cases} \end{aligned}$$

$$\begin{aligned} 16. \int \frac{dx}{a+b \sin x} &= \begin{cases} \frac{2}{\sqrt{a^2-b^2}} \arctan \frac{a \tan \frac{x}{2} + b}{\sqrt{a^2-b^2}} + C & (a^2 > b^2) \\ \frac{1}{\sqrt{b^2-a^2}} \ln \left| \frac{a \tan \frac{x}{2} + b - \sqrt{b^2-a^2}}{a \tan \frac{x}{2} + b + \sqrt{b^2-a^2}} \right| + C & (a^2 < b^2) \end{cases} \\ 17. \int \frac{dx}{a+b \cos x} &= \begin{cases} \frac{2}{a+b} \sqrt{\frac{a-b}{a+b}} \arctan \left(\sqrt{\frac{a-b}{a+b}} \tan \frac{x}{2} \right) + C & (a^2 > b^2) \\ \frac{1}{a+b} \sqrt{\frac{a+b}{a-b}} \ln \left| \frac{\tan \frac{x}{2} + \sqrt{\frac{a+b}{a-b}}}{\tan \frac{x}{2} - \sqrt{\frac{a+b}{a-b}}} \right| + C & (a^2 < b^2) \end{cases} \\ 18. \int \frac{dx}{a^2 \cos^2 x + b^2 \sin^2 x} &= \frac{1}{ab} \arctan \left(\frac{b}{a} \tan x \right) + C \\ 19. \int \frac{dx}{a^2 \cos^2 x - b^2 \sin^2 x} &= \frac{1}{2ab} \ln \left| \frac{b \tan x + a}{b \tan x - a} \right| + C \\ 20. \int x \sin ax dx &= -\frac{1}{a^2} \sin ax - \frac{1}{a} x \cos ax + C \\ 21. \int x^2 \sin ax dx &= -\frac{1}{a} x^2 \cos ax + \frac{2}{a^2} x \sin ax + \frac{2}{a^3} \cos ax + C \\ 22. \int x \cos ax dx &= \frac{1}{a^2} \cos ax + \frac{1}{a} x \sin ax + C \\ 23. \int x^2 \cos ax dx &= \frac{1}{a} x^2 \sin ax + \frac{2}{a^2} x \cos ax - \frac{2}{a^3} \sin ax + C \end{aligned}$$

反三角函数的积分 (其中 $a > 0$)

$$\begin{aligned} 1. \int \arcsin \frac{x}{a} dx &= x \arcsin \frac{x}{a} + \sqrt{a^2 - x^2} + C \\ 2. \int x \arcsin \frac{x}{a} dx &= \left(\frac{x^2}{2} - \frac{a^2}{4} \right) \arcsin \frac{x}{a} + \frac{x}{4} \sqrt{x^2 - x^2} + C \\ 3. \int x^2 \arcsin \frac{x}{a} dx &= \frac{x^3}{3} \arcsin \frac{x}{a} + \frac{1}{9} (x^2 + 2a^2) \sqrt{a^2 - x^2} + C \\ 4. \int \arccos \frac{x}{a} dx &= x \arccos \frac{x}{a} - \sqrt{a^2 - x^2} + C \end{aligned}$$

$$\begin{aligned} 5. \int x \arccos \frac{x}{a} dx &= \left(\frac{x^2}{2} - \frac{a^2}{4} \right) \arccos \frac{x}{a} - \frac{x}{4} \sqrt{a^2 - x^2} + C \\ 6. \int x^2 \arccos \frac{x}{a} dx &= \frac{x^3}{3} \arccos \frac{x}{a} - \frac{1}{9} (x^2 + 2a^2) \sqrt{a^2 - x^2} + C \\ 7. \int \arctan \frac{x}{a} dx &= x \arctan \frac{x}{a} - \frac{a}{2} \ln (a^2 + x^2) + C \\ 8. \int x \arctan \frac{x}{a} dx &= \frac{1}{2} (a^2 + x^2) \arctan \frac{x}{a} - \frac{a}{2} x + C \\ 9. \int x^2 \arctan \frac{x}{a} dx &= \frac{x^3}{3} \arctan \frac{x}{a} - \frac{a}{6} x^2 + \frac{a^3}{6} \ln (a^2 + x^2) + C \end{aligned}$$

指数函数的积分

$$\begin{aligned} 1. \int a^x dx &= \frac{1}{\ln a} a^x + C \\ 2. \int e^{ax} dx &= \frac{1}{a} e^{ax} + C \\ 3. \int x e^{ax} dx &= \frac{1}{a^2} (ax - 1) e^{ax} + C \\ 4. \int x^n e^{ax} dx &= \frac{1}{a} x^n e^{ax} - \frac{n}{a} \int x^{n-1} e^{ax} dx \\ 5. \int x a^x dx &= \frac{x}{\ln a} a^x - \frac{1}{(\ln a)^2} a^x + C \\ 6. \int x^n a^x dx &= \frac{1}{\ln a} x^n a^x - \frac{n}{\ln a} \int x^{n-1} a^x dx \\ 7. \int e^{ax} \sin bx dx &= \frac{1}{a^2+b^2} e^{ax} (a \sin bx - b \cos bx) + C \\ 8. \int e^{ax} \cos bx dx &= \frac{1}{a^2+b^2} e^{ax} (b \sin bx + a \cos bx) + C \\ 9. \int e^{ax} \sin^n bx dx &= \frac{1}{a^2+b^2} e^{ax} \sin^{n-1} bx (a \sin bx - nb \cos bx) + \frac{n(n-1)b^2}{a^2+b^2} \int e^{ax} \sin^{n-2} bx dx \\ 10. \int e^{ax} \cos^n bx dx &= \frac{1}{a^2+b^2} e^{ax} \cos^{n-1} bx (a \cos bx + nb \sin bx) + \frac{n(n-1)b^2}{a^2+b^2} \int e^{ax} \cos^{n-2} bx dx \\ 11. \int \ln x dx &= x \ln x - x + C \\ 12. \int \frac{dx}{x \ln x} &= \ln |\ln x| + C \\ 13. \int x^n \ln x dx &= \frac{1}{n+1} x^{n+1} (\ln x - \frac{1}{n+1}) + C \\ 14. \int (\ln x)^n dx &= x (\ln x)^n - n \int (\ln x)^{n-1} dx \\ 15. \int x^m (\ln x)^n dx &= \frac{1}{m+1} x^{m+1} (\ln x)^n - \frac{n}{m+1} \int x^m (\ln x)^{n-1} dx \end{aligned}$$

对数函数的积分

$$\begin{aligned} 1. \int \ln x dx &= x \ln x - x + C \\ 2. \int \frac{dx}{x \ln x} &= \ln |\ln x| + C \\ 3. \int x^n \ln x dx &= \frac{1}{n+1} x^{n+1} (\ln x - \frac{1}{n+1}) + C \\ 4. \int (\ln x)^n dx &= x (\ln x)^n - n \int (\ln x)^{n-1} dx \\ 5. \int x^m (\ln x)^n dx &= \frac{1}{m+1} x^{m+1} (\ln x)^n - \frac{n}{m+1} \int x^m (\ln x)^{n-1} dx \\ 6. \int_0^1 t^{x-1} (1-t)^{y-1} dt &= \operatorname{beta}(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} \\ 7. \int_0^{\infty} t^{num-1} e^{-t} dt &= \operatorname{gamma}(num) = \Gamma(num) \\ 8. \int_0^{\phi h i} \frac{d\theta}{\sqrt{1-k^2 \sin^2 \theta}} &= \operatorname{ellint}_1(k, phi) \\ 9. \int_0^{\phi h i} \sqrt{1-k^2 \sin^2 \theta} d\theta &= \operatorname{ellint}_2(k, phi) \\ 10. \int_{num}^{\infty} \frac{e^{-t}}{t} dt &= -\operatorname{expint}(-num) \\ 11. \sum_{n=1}^{\infty} n^{-num} &= \operatorname{riemann_zeta}(num) \\ 12. \int_0^{\infty} e^{-t^2} dt &= \operatorname{erf}(arg) \end{aligned}$$

STL 积分/求和 (need std::)

$$\begin{aligned} 1. \int_0^1 t^{x-1} (1-t)^{y-1} dt &= \operatorname{beta}(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)} \\ 2. \int_0^{\infty} t^{num-1} e^{-t} dt &= \operatorname{gamma}(num) = \Gamma(num) \\ 3. \int_0^{\phi h i} \frac{d\theta}{\sqrt{1-k^2 \sin^2 \theta}} &= \operatorname{ellint}_1(k, phi) \\ 4. \int_0^{\phi h i} \sqrt{1-k^2 \sin^2 \theta} d\theta &= \operatorname{ellint}_2(k, phi) \\ 5. \int_{num}^{\infty} \frac{e^{-t}}{t} dt &= -\operatorname{expint}(-num) \\ 6. \sum_{n=1}^{\infty} n^{-num} &= \operatorname{riemann_zeta}(num) \\ 7. \int_0^{\infty} e^{-t^2} dt &= \operatorname{erf}(arg) \end{aligned}$$