Team Reference Document

Heltion

2024 年 10 月 6 日

目录

1 Template

1.1 .clang-format

```
1  BasedOnStyle: Google
2  IndentWidth: 4
3  ColumnLimit: 160
```

1.2 debug.cpp

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  template <class T, size_t size = tuple_size<T>::value>
6  string to_debug(T, string s = "") requires(not ranges::range<T>);
7
8  template<class T>
9  concept check = requires(T x, ostream &os) {
10   os << x;
11 };
12
13 template<check T>
14 string to_debug(T x) {
15   return static_cast<ostringstream>(ostringstream() << x).str();
16 }
17
18 string to_debug(ranges::range auto x, string s = "") requires(not is_same_v<
       decltype(x), string>) {
19   for (auto xi : x) {
20     s += ",␣" + to_debug(xi);
21   }
22   return "[␣" + s.substr(s.empty() ? 0 : 2) + "␣]";
23 }
24
25 template <class T, size_t size>
26 string to_debug(T x, string s) requires(not ranges::range<T>) {
27   [&]<size_t... I>(index_sequence<I...>) {
28     ((s += ",␣" + to_debug(get<I>(x))), ...);
29   }(make_index_sequence<size>());
30   return "{" + s.substr(s.empty() ? 0 : 2) + "}";
31 }
32
33 #define debug(...) [](auto... $){ ((cout << to_debug($) << "␣"), ...); cout
       << endl; }("[", #__VA_ARGS__, "]:", __VA_ARGS__)
```

1.3 gen.py

```
1  from random import *
2  n = 10000
3  s = 'qwertyuiopasdfghjklzxcvbnm'
4  for i in range(n):
5      print(choice(s), end = '')
6  print()
7  print(randint(0, 1), randint(1, n))
```

1.4 head.cpp

```
1  #pragma GCC optimize("Ofast", "inline", "unroll-loops")
2  #include <bits/stdc++.h>
3  using namespace std;
4  #define rep(i, a, n) for (int i = a; i <= n; i++)
5  #define per(i, a, n) for (int i = a; i >= n; i--)
6  #define pb push_back
7  #define eb emplace_back
8  #define all(x) (x).begin(), (x).end()
```

```
 9   #define bit(x) (1ll << (x))
10   #define fi first
11   #define se second
12   #define SZ(x) ((int)(x).size())
13   using VI = vector<int>;
14   using PII = pair<int, int>;
15   using ll = long long;
16   using ull = unsigned long long;
17   using db = double;
18   using ldb = long double;
19   mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
20   // head
21
22   #ifdef DEBUG
23   #include "debug.cpp"
24   #else
25   #define debug(...) 42
26   #endif
27
28   void solve() {}
29   int main() {
30       cin.tie(nullptr)->sync_with_stdio(false);
31       cout << fixed << setprecision(16);
32       int tt = 1;
33       cin >> tt;
34       while (tt--) {
35           solve();
36       }
37   }
```

1.5   head-apiadu.cpp

```
 1   #include <bits/stdc++.h>
 2   using namespace std;
 3   #define rep(i,a,n) for (int i=a;i<n;i++)
 4   #define per(i,a,n) for (int i=n-1;i>=a;i--)
 5   #define pb push_back
 6   #define eb emplace_back
 7   #define mp make_pair
 8   #define all(x) (x).begin(),(x).end()
 9   #define fi first
10   #define se second
11   #define SZ(x) ((int)(x).size())
```

```
12   typedef vector<int> VI;
13   typedef basic_string<int> BI;
14   typedef long long ll;
15   typedef pair<int,int> PII;
16   typedef double db;
17   mt19937 mrand(random_device{}());
18   const ll mod=1000000007;
19   int rnd(int x) { return mrand() % x;}
20   ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)
         res=res*a%mod;a=a*a%mod;}return res;}
21   ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}
22   // head
```

1.6   Makefile

```
 1   %: %.cpp
 2       g++-13 $< -o $@ -std=gnu++20 -g -O2 -Wall -Wextra -DDEBUG -
            D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
```

1.7   pai.py

```
 1   import os
 2   tt = 0
 3   while True:
 4       os.system('python gen.py > A.in')
 5       os.system('./a < A.in > a.out')
 6       os.system('./b < A.in > b.out')
 7       # diff for linux or macos, fc for windows
 8       if os.system('diff a.out b.out'):
 9           print("WA")
10           exit(0)
11       else:
12           tt += 1
13           print("AC:", tt)
```

1.8   settings.json

```
 1   {
 2       "editor.formatOnSave": true,
 3       "files.autoSave": "afterDelay",
 4       "files.autoSaveDelay": 350,
 5       "C_Cpp.default.cppStandard": "gnu++20"
 6   }
```

2 Data

2.1 01trie.cpp

```
1  struct node {
2      int son[2];
3      int end;
4      int sz;
5  } seg[maxn << 2];
6  int root, tot;
7  int n, m;
8
9  void insert(ll x) {
10     int cnt = root;
11     for (int i = 62; i >= 0; i--) {
12         int w = (x >> i) & 1;
13         if (seg[cnt].son[w] == 0) seg[cnt].son[w] = ++tot;
14         cnt = seg[cnt].son[w];
15         seg[cnt].sz++;
16     }
17     seg[cnt].end++;
18 }
19
20 ll query(ll x, ll k) {
21     ll res = 0;
22     int cnt = root;
23     for (int i = 62; i >= 0; i--) {
24         int w = (x >> i) & 1;
25         if (seg[seg[cnt].son[w]].sz >= k) cnt = seg[cnt].son[w];
26         else {
27             k -= seg[seg[cnt].son[w]].sz;
28             cnt = seg[cnt].son[abs(w - 1)];
29             res += bit(i);
30         }
31     }
32     return res;
33 }
```

2.2 2dtree(bqi343).cpp

```
1  const int SZ = 1.1e5;
2  template <class T>
3  struct node {
```

```
4      T val = 0;
5      node<T>* c[2];
6      node() { c[0] = c[1] = NULL; }
7      void upd(int ind, T v, int L = 0, int R = SZ - 1) {  // add v
8          if (L == ind && R == ind) {
9              val += v;
10             return;
11         }
12         int M = (L + R) / 2;
13         if (ind <= M) {
14             if (!c[0]) c[0] = new node();
15             c[0]->upd(ind, v, L, M);
16         } else {
17             if (!c[1]) c[1] = new node();
18             c[1]->upd(ind, v, M + 1, R);
19         }
20         val = 0;
21         rep(i, 0, 1) if (c[i]) val += c[i]->val;
22     }
23     T query(int lo, int hi, int L = 0, int R = SZ - 1) {  // query sum of
           segment
24         if (hi < L || R < lo) return 0;
25         if (lo <= L && R <= hi) return val;
26         int M = (L + R) / 2;
27         T res = 0;
28         if (c[0]) res += c[0]->query(lo, hi, L, M);
29         if (c[1]) res += c[1]->query(lo, hi, M + 1, R);
30         return res;
31     }
32     void UPD(int ind, node* c0, node* c1, int L = 0, int R = SZ - 1) {  //
           for 2D segtree
33         if (L != R) {
34             int M = (L + R) / 2;
35             if (ind <= M) {
36                 if (!c[0]) c[0] = new node();
37                 c[0]->UPD(ind, c0 ? c0->c[0] : NULL, c1 ? c1->c[0] : NULL, L
                     , M);
38             } else {
39                 if (!c[1]) c[1] = new node();
40                 c[1]->UPD(ind, c0 ? c0->c[1] : NULL, c1 ? c1->c[1] : NULL, M
                     + 1, R);
41             }
42         }
```

```
43            val = (c0 ? c0->val : 0) + (c1 ? c1->val : 0);
44        }
45 };
46
47 /**
48  * Description: BIT of SegTrees. $x\in (0,SZ), y\in [0,SZ)$.
49  * Memory: O(N\log~2 N)
50  * Source: USACO Mowing the Field
51  * Verification:
52  * USACO Mowing the Field
53  * http://www.usaco.org/index.php?page=viewproblem2&cpid=722 (13/15, 15/15
        and 1857ms with BumpAllocator)
54  */
55
56 #include "../1D␣Range␣Queries␣(9.2)/SparseSeg␣(9.2).h"
57
58 template <class T>
59 struct BITseg {
60     node<T> seg[SZ];
61     BITseg() { fill(seg, seg + SZ, node<T>()); }
62     void upd(int x, int y, int v) {  // add v
63        for (; x < SZ; x += x & -x) seg[x].upd(y, v);
64     }
65     T query(int x, int yl, int yr) {
66        T res = 0;
67        for (; x; x -= x & -x) res += seg[x].query(yl, yr);
68        return res;
69     }
70     T query(int xl, int xr, int yl, int yr) {  // query sum of rectangle
71        return query(xr, yl, yr) - query(xl - 1, yl, yr);
72     }
73 };
74
75 /**
76  * Description: SegTree of SegTrees. $x,y\in [0,SZ).$
77  * Memory: O(N\log~2 N)
78  * Source: USACO Mowing the Field
79  * Verification:
80  * http://www.usaco.org/index.php?page=viewproblem2&cpid=722 (9/15 w/
        BumpAllocator)
81  * http://www.usaco.org/index.php?page=viewproblem2&cpid=601 (4238 ms, 2907
        ms w/ BumpAllocator)
82  */
83
84 #include "../1D␣Range␣Queries␣(9.2)/SparseSeg␣(9.2).h"
85
86 template <class T>
87 struct Node {
88     node<T> seg;
89     Node* c[2];
90     Node() { c[0] = c[1] = NULL; }
91     void upd(int x, int y, T v, int L = 0, int R = SZ - 1) {  // add v
92        if (L == x && R == x) {
93            seg.upd(y, v);
94            return;
95        }
96        int M = (L + R) / 2;
97        if (x <= M) {
98            if (!c[0]) c[0] = new Node();
99            c[0]->upd(x, y, v, L, M);
100       } else {
101           if (!c[1]) c[1] = new Node();
102           c[1]->upd(x, y, v, M + 1, R);
103       }
104       seg.upd(y, v);  // only for addition
105       // seg.UPD(y,c[0]?&c[0]->seg:NULL,c[1]?&c[1]->seg:NULL);
106    }
107    T query(int x1, int x2, int y1, int y2, int L = 0, int R = SZ - 1) {  //
           query sum of rectangle
108       if (x1 <= L && R <= x2) return seg.query(y1, y2);
109       if (x2 < L || R < x1) return 0;
110       int M = (L + R) / 2;
111       T res = 0;
112       if (c[0]) res += c[0]->query(x1, x2, y1, y2, L, M);
113       if (c[1]) res += c[1]->query(x1, x2, y1, y2, M + 1, R);
114       return res;
115    }
116 };
```

2.3 cdq.cpp

```
1 int ans[maxn], lev[maxn];
2 array<int, 5> v[maxn], tmp[maxn];
3
4 struct BIT {
5        ...
```

```
6  } c;
7
8  void solve(int l, int r) {
9      if (l >= r) return;
10     int mid = (l + r) / 2;
11     solve(l, mid), solve(mid + 1, r);
12     int i = l, j = mid + 1;
13     int piv = l;
14     while (i <= mid || j <= r) {
15         if (i <= mid && (j > r || mp(v[i][1], v[i][2]) <= mp(v[j][1], v[j
               ][2]))) {
16             c.modify(v[i][2], v[i][3]);
17             tmp[piv++] = v[i++];
18         } else {
19             v[j][4] += c.query(v[j][2]);
20             tmp[piv++] = v[j++];
21         }
22     }
23     rep(i, l, mid) c.modify(v[i][2], -v[i][3]);
24     rep(i, l, r) v[i] = tmp[i];
25 }
26
27 void solve() {
28     cin >> n >> k;
29     c.resize(k);
30     rep(i, 1, n) {
31         int s, c, m;
32         cin >> s >> c >> m;
33         v[i] = {s, c, m, 1, 0};
34     }
35     v[0][0] = -1;
36     sort(v + 1, v + n + 1);
37     int cnt = 0;
38     rep(i, 1, n) {
39         if (v[i][0] == v[cnt][0] && v[i][1] == v[cnt][1] && v[i][2] == v[cnt
               ][2]) v[cnt][3]++;
40         else v[++cnt] = v[i];
41     }
42     solve(1, cnt);
43     rep(i, 1, cnt) {
44         ans[v[i][4] + v[i][3] - 1] += v[i][3];
45     }
46     rep(i, 0, n - 1) cout << ans[i] << '\n';
```

2.4 compact.cpp

```
1  namespace compact {
2  const int LOGN=18;
3  int l[N],r[N],tot,p[N][20],n;
4  map<int,int> cv;
5  int lca(int u,int v) {
6      if (dep[u]>dep[v]) swap(u,v);
7      per(i,LOGN-1,0) if (dep[p[v][i]]>=dep[u]) v=p[v][i];
8      if (u==v) return u;
9      per(i,LOGN-1,0) if (p[v][i]!=p[u][i]) u=p[u][i],v=p[v][i];
10     return p[u][0];
11 }
12 void dfs(int u,int f) {
13     l[u]=++tot; dep[u]=dep[f]+1; p[u][0]=f;
14     vec[dep[u]].pb(u);
15     for (auto v:vE[u]) {
16         if (v==f) continue;
17         dfs(v,u);
18     }
19     r[u]=tot;
20 }
21 void build(int _n) {
22     n=_n; tot=0;
23     dfs(1,0);
24     rep(j,1,LOGN-1) rep(i,1,n) p[i][j]=p[p[i][j-1]][j-1];
25 }
26
27 bool cmp(int u,int v) { return l[u]<l[v]; }
28 vector<PII> compact(VI v) {
29     int m=SZ(v);
30     vector<PII> E;
31     sort(all(v),cmp);
32     rep(i,0,m-2) {
33         int w=lca(v[i],v[i+1]);
34         v.pb(w);
35     }
36     v.pb(0);
37     v.pb(1);
38     sort(all(v),cmp);
39     v.erase(unique(all(v)),v.end());
```

```
40      cv.clear();
41      per(i,SZ(v)-1,1) {
42          int u=v[i];
43          while (1) {
44              auto it=cv.lower_bound(l[u]);
45              if (it==cv.end()||it->fi>r[u]) break;
46              E.pb(mp(u,v[it->se]));
47              cv.erase(it);
48          }
49          cv[l[u]]=i;
50      }
51      return E;
52 }
53 };
```

2.5   dominator.cpp

```
1  void solve(int u, int S) {
2    int best = -1, cnt = S + 1;
3    auto find_best = [&](auto &find_best, int u, int par) -> void {
4      sz[u] = 1, sdom[u] = 0;
5      for (auto v : e[u]) {
6        if (v == par || del[v]) continue;
7        find_best(find_best, v, u);
8        sz[u] += sz[v];
9        sdom[u] = max(sdom[u], sz[v]);
10     }
11     sdom[u] = max(sdom[u], S - sz[u]);
12     if (sdom[u] < cnt) {
13       cnt = sdom[u], best = u;
14     }
15   };
16   find_best(find_best, u, 0);
17   int id1 = tot++, dep1 = 0;
18   int id2, dep2;
19   auto dfs = [&](auto &dfs, int u, int par, int dep) -> void {
20     dep1 = max(dep1, dep);
21     dep2 = max(dep2, dep);
22     Q[u].pb({id1, 1, dep});
23     Q[u].pb({id2, -1, dep});
24     for (auto v : e[u]) {
25       if (v == par || del[v]) continue;
26       dfs(dfs, v, u, dep + 1);
```

```
27     }
28   };
29   Q[best].pb({id1, 1, 0});
30   for (auto v : e[best]) {
31     if (del[v]) continue;
32     id2 = tot++, dep2 = 0;
33     dfs(dfs, v, best, 1);
34     fenw[id2] = BIT<ll>(dep2 + 1);
35   }
36   fenw[id1] = BIT<ll>(dep1 + 1);
37   del[best] = 1;
38   for (auto v : e[best]) {
39     if (!del[v]) solve(v, sz[v]);
40   }
41 }
```

2.6   dsu.cpp

```
1  class dsu {
2   public:
3    vector<int> fa;
4    vector<ll> dist;
5    int n;
6
7    dsu(int _n) : n(_n) {
8      fa.resize(n);
9      dist.assign(n, 0);
10     iota(fa.begin(), fa.end(), 0);
11   }
12
13   int find(int x) {
14     if (fa[x] == x) return x;
15     int par = fa[x];
16     fa[x] = find(fa[x]);
17     dist[x] += dist[par];
18     return fa[x];
19   }
20
21   void unite(int x, int y, ll v) {
22     int px = find(x);
23     int py = find(y);
24     fa[py] = px;
25     dist[py] = dist[x] - dist[y] - v;
```

```
26    }
27 };
```

2.7   dsu-on-tree.cpp

```
1  void dfs(int x, int fa) {
2      hs[x] = -1, w[x] = 1;
3      l[x] = ++tot;
4      id[tot] = x;
5      for (auto y : g[x]) if (y != fa) {
6              dfs(y, x);
7              w[x] += w[y];
8              if (hs[x] == -1 || w[y] > w[hs[x]])
9                  hs[x] = y;
10         }
11     r[x] = tot;
12 }
13
14 void dsu(int x, int fa, int keep) {
15     for (auto y : g[x]) {
16         if (y != hs[x] && y != fa) {
17             dsu(y, x, 0);
18         }
19     }
20     if (hs[x] != -1) dsu(hs[x], x, 1);
21
22     for (auto y : g[x]) {
23         if (y != hs[x] && y != fa) {
24             for (int i = l[y]; i <= r[y]; i++) {
25
26             }
27         }
28     }
29     // add current node
30
31     ans[x] = cnt;
32
33     if (!keep) {
34         // clear
35     }
36 }
```

2.8   fenwick.cpp

```
1  template <typename T>
2  struct BIT {
3    vector<T> fenw;
4    int n, pw;
5
6    BIT(int n_ = 0) : n(n_) {
7      fenw.assign(n + 1, 0);
8      pw = bit_floor(unsigned(n));
9    }
10
11   void Modify(int x, T v) {
12     if (x <= 0) return;  // assert(0 <= x && x < n);
13     while (x <= n) {       // x < n
14       fenw[x] += v;
15       x += (x & -x);  // x |= x + 1;
16     }
17   }
18
19   T Query(int x) {
20     // assert(0 <= x && x <= n);
21     T v{};
22     while (x > 0) {
23       v += fenw[x];   // fenw[x - 1];
24       x -= (x & -x);  // x &= x - 1;
25     }
26     return v;
27   }
28
29   // Returns the length of the longest prefix with sum <= c
30   int MaxPrefix(T c) {
31     T v{};
32     int at = 0;
33     for (int i = 20; i >= 0; i--) {
34       if (at + bit(i) <= n && v + fenw[at + bit(i)] <= c) {
35         v += fenw[at + bit(i)];
36         at += bit(i);
37       }
38     }
39     /**
40      * for (int len = pw; len > 0; len >>= 1) {
41      *   if (at + len <= n) {
42      *     auto nv = v;
```

```
43   *        nv += fenw[at + len - 1];
44   *        if (!(c < nv)) {
45   *          v = nv;
46   *          at += len;
47   *        }
48   *      }
49   * }
50   * assert(0 <= at && at <= n);
51   */
52     return at;
53   }
54 };
```

2.9 fenwick2d.cpp

```
1  template <typename T>
2  class fenwick2d {
3   public:
4    vector<vector<T>> fenw;
5    int n, m;
6
7    fenwick2d(int _n, int _m) : n(_n), m(_m) {
8      fenw.resize(n);
9      for (int i = 0; i < n; i++) {
10       fenw[i].resize(m);
11     }
12   }
13
14   inline void modify(int i, int j, T v) {
15     int x = i;
16     while (x < n) {
17       int y = j;
18       while (y < m) {
19         fenw[x][y] += v;
20         y |= (y + 1);
21       }
22       x |= (x + 1);
23     }
24   }
25
26   inline T get(int i, int j) {
27     T v{};
28     int x = i;
```

```
29     while (x >= 0) {
30       int y = j;
31       while (y >= 0) {
32         v += fenw[x][y];
33         y = (y & (y + 1)) - 1;
34       }
35       x = (x & (x + 1)) - 1;
36     }
37     return v;
38   }
39 };
40
41 struct node {
42   int a = ...;  // don't forget to set default value
43
44   inline void operator+=(node &other) { ... }
45 };
```

2.10 hash-table.cpp

```
1  struct Hash_table {
2      static const int V = 1000003;
3      int fst[V], nxt[V];
4      int ctm, ptm[V], T;
5      int val[V];
6      ll key[V];
7      void init() {T = 0, ctm++;}
8      void insert(ll k, int v) {
9          int s = k % V;
10         if (ptm[s] != ctm) ptm[s] = ctm, fst[s] = -1;
11         for (int i = fst[s]; i != -1; i = nxt[i]) if (key[i] == k) {
12                 return;
13             }
14         nxt[T] = fst[s], fst[s] = T, key[T] = k, val[T] = v;
15         T++;
16     }
17     int query(ll k) {
18         int s = k % V;
19         if (ptm[s] != ctm) return -1;
20         for (int i = fst[s]; i != -1; i = nxt[i]) {
21             if (key[i] == k) return val[i];
22         }
23         return -1;
```

```
24        }
25    };
```

2.11 hilbertOrder.cpp

```
 1   template <class M, bool val_on_edge = false>
 2   class Mo_Tree {
 3   private:
 4       static inline int64_t hilbertOrder(int x, int y, int pow, int rotate) {
 5           if (pow == 0)
 6               return 0;
 7           int hpow = 1 << (pow - 1);
 8           int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) : ((y < hpow) ? 1 : 2);
 9           seg = (seg + rotate) & 3;
10           const int rotateDelta[4] = {3, 0, 0, 1};
11           int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
12           int nrot = (rotate + rotateDelta[seg]) & 3;
13           int64_t subSquareSize = int64_t(1) << (2 * pow - 2);
14           int64_t ordd = seg * subSquareSize;
15           int64_t add = hilbertOrder(nx, ny, pow - 1, nrot);
16           ordd += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
17           return ordd;
18       }
19
20       struct query {
21           int l, r, q_indx, lca;
22           int64_t ord;
23       };
24
25       int q;
26       vector<int> depth, flat, st, en, node_freq;
27       vector<vector<int>> anc;
28       vector<M> par_v;
29       vector<query> Q;
30
31       void add_node(int node) {
32           if (par_v[node] < N) {
33               aux[par_v[node]]++;
34               w.reset(par_v[node]);
35           }
36       }
37
38       void remove_node(int node) {
```

```
39           if (par_v[node] < N) {
40               if (--aux[par_v[node]] == 0) {
41                   w.set(par_v[node]);
42               }
43           }
44       }
45
46       long long calc(...) {
47           return w._Find_first();
48       }
49
50       void add(int ind) {
51           node_freq[flat[ind]]++;
52           if (node_freq[flat[ind]] == 1) {
53               add_node(flat[ind]);
54           } else {
55               remove_node(flat[ind]);
56           }
57       }
58
59       void remove(int ind) {
60           node_freq[flat[ind]]--;
61           if (node_freq[flat[ind]] == 1) {
62               add_node(flat[ind]);
63           } else {
64               remove_node(flat[ind]);
65           }
66       }
67
68   public:
69       Mo_Tree(vector<vector<int>> &g, const vector<M> &v) {
70           q = 0;
71           this->par_v = v;
72           int n = (int)g.size() - 1;
73           depth.resize(n + 1), st.resize(n + 1);
74           en.resize(n + 1), node_freq.resize(n + 1);
75           anc.assign(n + 1, vector<int>(__lg(n) + 2, 0));
76
77           auto dfs = [&](auto &dfs, int node, int p, int d) -> void {
78               anc[node][0] = p;
79               depth[node] = d;
80               st[node] = flat.size();
81               flat.push_back(node);
```

```cpp
82              for (auto it : g[node]) {
83                  if (it != p) {
84                      dfs(dfs, it, node, d + 1);
85                  }
86              }
87              en[node] = flat.size();
88              flat.push_back(node);
89          };
90          dfs(dfs, 1, 0, 0);
91
92          for (int j = 1; j <= __lg(n); ++j) {
93              for (int i = 1; i <= n; ++i) {
94                  anc[i][j] = anc[anc[i][j - 1]][j - 1];
95              }
96          }
97      }
98
99      int kth_anc(int node, int k) {
100         int ret = node;
101         for (int bit = (int)anc[ret].size() - 1; ~bit; --bit) {
102             if (k & (1 << bit)) {
103                 ret = anc[ret][bit];
104             }
105         }
106         return ret;
107     }
108
109     int LCA(int u, int v) {
110         if (depth[u] < depth[v]) {
111             swap(u, v);
112         }
113         u = kth_anc(u, depth[u] - depth[v]);
114         if (u == v) {
115             return u;
116         }
117         for (int bit = anc[0].size() - 1; ~bit; --bit) {
118             if (anc[u][bit] != anc[v][bit]) {
119                 u = anc[u][bit];
120                 v = anc[v][bit];
121             }
122         }
123         return anc[u][0];
124     }
```

```cpp
125
126     template <class... T>
127     void add_query(int u, int v, T &...x) {
128         if (st[u] > st[v]) {
129             swap(u, v);
130         }
131         int lca = LCA(u, v), l, r, q_lca = -1;
132         if (lca == u || lca == v) {
133             l = st[u] + val_on_edge, r = st[v];
134         } else {
135             l = en[u], r = st[v], q_lca = lca;
136         }
137         Q.push_back({l, r, q++, q_lca, hilbertOrder(l, r, __lg(flat.size())
138             + 1, 0), x...});
139     }
140
141     vector<int> Mo() {
142         vector<int> ans(q);
143         sort(Q.begin(), Q.end(), [&](const query &a, const query &b) {
144             return a.ord < b.ord;
145         });
146         int l = 0, r = -1;
147         for (auto [L, R, q_indx, lca, ord] : Q) {
148             while (l > L)
149                 add(--l);
150             while (r < R)
151                 add(++r);
152             while (r > R)
153                 remove(r--);
154             while (l < L)
155                 remove(l++);
156
157             if (~lca && !val_on_edge)
158                 add_node(lca);
159             ans[q_indx] = calc();
160             if (~lca && !val_on_edge)
161                 remove_node(lca);
162         }
163         return ans;
164     }
165 };
```

2.12 HLD.cpp

```cpp
1   struct HLD {
2       int n;
3       std::vector<int> siz, top, dep, parent, in, out, seq;
4       std::vector<std::vector<int>> adj;
5       int cur;
6
7       HLD() {}
8       HLD(int n) {
9           init(n);
10      }
11      void init(int n) {
12          this->n = n;
13          siz.resize(n);
14          top.resize(n);
15          dep.resize(n);
16          parent.resize(n);
17          in.resize(n);
18          out.resize(n);
19          seq.resize(n);
20          cur = 0;
21          adj.assign(n, {});
22      }
23      void addEdge(int u, int v) {
24          adj[u].push_back(v);
25          adj[v].push_back(u);
26      }
27      void work(int root = 0) {
28          top[root] = root;
29          dep[root] = 0;
30          parent[root] = -1;
31          dfs1(root);
32          dfs2(root);
33      }
34      void dfs1(int u) {
35          if (parent[u] != -1) {
36              adj[u].erase(std::find(adj[u].begin(), adj[u].end(), parent[u]))
                        ;
37          }
38
39          siz[u] = 1;
40          for (auto &v : adj[u]) {
41              parent[v] = u;
```

```cpp
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 isAncester(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 (!isAncester(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 (!isAncester(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 };
```

2.13   kdtree.cpp

```
1  namespace kd {
2  const int K = 2, N = 2.1e5;
3  template <typename T>
4  using P = array<T, K>;
```

```
5   template <typename T>
6   struct node {
7     P<T> pt, mx, mn;
8     ll val, sum;
9     node *l, *r, *p;
10    int id;
11    node(const P<T> &_pt = P<T>(), ll _val = 0, int _id = 0)
12        : pt(_pt), val(_val), sum(_val), id(_id) {
13      mx = mn = pt;
14      p = l = r = nullptr;
15    }
16  };
17  node<ll> *ptr[N];
18  template <typename T>
19  void pull(node<T> *u) {
20    if (not u) return;
21    u->sum = u->val;
22    rep(i, 0, K - 1) u->mx[i] = u->mn[i] = u->pt[i];
23    if (u->l) {
24      u->sum += u->l->sum;
25      u->l->p = u;
26    }
27    if (u->r) {
28      u->sum += u->r->sum;
29      u->r->p = u;
30    }
31    rep(i, 0, K - 1) {
32      if (u->l) {
33        u->mx[i] = max(u->mx[i], u->l->mx[i]);
34        u->mn[i] = min(u->mn[i], u->l->mn[i]);
35      }
36      if (u->r) {
37        u->mx[i] = max(u->mx[i], u->r->mx[i]);
38        u->mn[i] = min(u->mn[i], u->r->mn[i]);
39      }
40    }
41  }
42
43  template <typename T>
44  node<T> *build(vector<node<T>> &a, int l, int r, int d = 0) {
45    if (d == K) d = 0;
46    if (l >= r) {
47      return nullptr;
```

```cpp
48      } else {
49        int md = (l + r) >> 1;
50        nth_element(a.begin() + l, a.begin() + md, a.begin() + r,
51                    [&](node<T> &x, node<T> &y) { return x.pt[d] < y.pt[d]; });
52        node<T> *p = new node<T>(a[md]);
53        ptr[p->id] = p;
54        p->l = build(a, l, md, d + 1);
55        p->r = build(a, md + 1, r, d + 1);
56        pull(p);
57        return p;
58      }
59    }
60
61    template <typename T>
62    node<T> *search(node<T> *u, P<T> p, int d = 0) {
63      if (d == K) d = 0;
64      if (not u) return nullptr;
65      if (u->pt == p) return u;
66      if (p[d] < u->pt[d]) {
67        return search(u->l, p, d + 1);
68      } else if (p[d] > u->pt[d]) {
69        return search(u->r, p, d + 1);
70      } else {
71        auto tmp = search(u->l, p, d + 1);
72        if (tmp) return tmp;
73        return search(u->r, p, d + 1);
74      }
75    }
76
77    template <typename T>
78    void modify(node<T> *u, ll v) {
79      if (not u) return;
80      u->val = v;
81      for (auto cur = u; cur; cur = cur->p) {
82        pull(cur);
83      }
84    }
85
86    template <typename T>
87    bool inside(node<T> *nd, P<T> p, ll c) {
88      int cc = 0;
89      if (nd->mx[0] * p[0] + nd->mx[1] * p[1] >= c) cc++;
90      if (nd->mn[0] * p[0] + nd->mn[1] * p[1] >= c) cc++;
91      if (nd->mx[0] * p[0] + nd->mn[1] * p[1] >= c) cc++;
92      if (nd->mn[0] * p[0] + nd->mx[1] * p[1] >= c) cc++;
93      return cc == 0;
94    }
95
96    template <typename T>
97    bool outside(node<T> *nd, P<T> p, ll c) {
98      int cc = 0;
99      if (nd->mx[0] * p[0] + nd->mx[1] * p[1] >= c) cc++;
100       if (nd->mn[0] * p[0] + nd->mn[1] * p[1] >= c) cc++;
101       if (nd->mx[0] * p[0] + nd->mn[1] * p[1] >= c) cc++;
102       if (nd->mn[0] * p[0] + nd->mx[1] * p[1] >= c) cc++;
103       return cc == 4;
104    }
105
106    template <typename T>
107    ll query(node<T> *u, P<T> p, ll c) {
108      if (inside(u, p, c)) return u->sum;
109      if (outside(u, p, c)) return 0;
110      ll s = 0;
111      if (u->pt[0] * p[0] + u->pt[1] * p[1] < c) {
112        s += u->val;
113      }
114      if (u->l) s += query(u->l, p, c);
115      if (u->r) s += query(u->r, p, c);
116      return s;
117    }
118
119    template <typename T>
120    T eval_min(node<T> *nd,
121               P<T> p) {  // 通过估价函数进行启发式搜索，根据当前结果对搜索剪枝
122      if (not nd) return numeric_limits<T>::max() / 4;
123      ll s = 0;
124      rep(i, 0, K - 1) {
125        if (p[i] <= nd->mn[i]) s += nd->mn[i] - p[i];
126        if (p[i] >= nd->mx[i]) s += p[i] - nd->mx[i];
127      }
128      return s;
129    }
130
131    template <typename T>
132    ll mindist(node<T> *u, P<T> p) {
133      ll s = numeric_limits<T>::max() / 4;
```

```
134     if (u->pt != p) {
135       s = min(s, abs(u->pt[0] - p[0]) + abs(u->pt[1] - p[1]));
136     }
137     ll best1 = eval_min(u->l, p), best2 = eval_min(u->r, p);
138     if (best1 < best2) {
139       if (u->l) s = min(s, mindist(u->l, p));
140       if (u->r and best2 < s) s = min(s, mindist(u->r, p));
141       return s;
142     } else {
143       if (u->r) s = min(s, mindist(u->r, p));
144       if (u->l and best1 < s) s = min(s, mindist(u->l, p));
145       return s;
146     }
147   }
148
149   template <typename T>
150   T eval_max(node<T> *nd,
151             P<T> p) {  // 通过估价函数进行启发式搜索, 根据当前结果对搜索剪枝
152     if (not nd) return 0;
153     ll s = 0;
154     rep(i, 0, K - 1) s += max(abs(nd->mx[i] - p[i]), abs(nd->mn[i] - p[i]));
155     return s;
156   }
157
158   template <typename T>
159   ll maxdist(node<T> *u, P<T> p) {
160     ll s = 0;
161     if (u->pt != p) {
162       s = max(s, abs(u->pt[0] - p[0]) + abs(u->pt[1] - p[1]));
163     }
164     ll best1 = eval_max(u->l, p), best2 = eval_max(u->r, p);
165     if (best1 > best2) {
166       if (u->l) s = max(s, maxdist(u->l, p));
167       if (u->r and best2 > s) s = max(s, maxdist(u->r, p));
168       return s;
169     } else {
170       if (u->r) s = max(s, maxdist(u->r, p));
171       if (u->l and best1 > s) s = max(s, maxdist(u->l, p));
172       return s;
173     }
174   }
175   }  // namespace kd
```

2.14 LCT.cpp

```
1  namespace linkCutTree {
2  struct node {
3      node *child[2], *parent, *max;
4      int id;
5      ll sum, val, sz, weight, rev;
6      node(ll val, ll weight, int id) : child {nullptr, nullptr}, parent(
7          nullptr), max(this), sum(val), val(val), sz(weight), weight(weight),
8          id(id), rev(false) {}
9  };
8
9  bool isRoot(node *p) { return p->parent == nullptr || (p->parent->child[0]
10     != p && p->parent->child[1] != p); }
10
11 int side(node *p) { return p->parent->child[1] == p; }
12
13 ll sum(node *p) { return p == nullptr ? 0 : p->sum; }
14
15 ll sz(node *p) { return p == nullptr ? 0 : p->sz; }
16
17 node *max(node *p) { return p == nullptr ? nullptr : p->max; }
18
19 node *max(node *p, node *q) {
20     if (p == nullptr)
21         return q;
22     if (q == nullptr)
23         return p;
24     return p->weight > q->weight ? p : q;
25 }
26
27 void reverse(node *p) {
28     if (p == nullptr)
29         return;
30     swap(p->child[0], p->child[1]);
31     p->rev ^= 1;
32 }
33
34 void push(node *p) {
35     if (p->rev == 0)
36         return;
37     p->rev = 0;
38     reverse(p->child[0]);
39     reverse(p->child[1]);
```

```
40  }
41
42  void pull(node *p) {
43      p->sum = sum(p->child[0]) + sum(p->child[1]) + p->val;
44      p->max = max(max(max(p->child[0]), max(p->child[1])), p);
45      p->sz = p->weight + sz(p->child[0]) + sz(p->child[1]);
46  }
47
48  void connect(node *p, node *q, int side) {
49      q->child[side] = p;
50      if (p != nullptr)
51          p->parent = q;
52  }
53
54  void rotate(node *p) {
55      auto q = p->parent;
56      int dir = side(p) ^ 1;
57      connect(p->child[dir], q, dir ^ 1);
58      if (!isRoot(q))
59          connect(p, q->parent, side(q));
60      else
61          p->parent = q->parent;
62      connect(q, p, dir);
63      pull(q);
64  }
65
66  void splay(node *p) {
67      vector<node *> stk;
68      for (auto i = p; !isRoot(i); i = i->parent)
69          stk.push_back(i->parent);
70      while (!stk.empty()) {
71          push(stk.back());
72          stk.pop_back();
73      }
74      push(p);
75      while (!isRoot(p)) {
76          auto q = p->parent;
77          if (!isRoot(q))
78              rotate(side(p) == side(q) ? q : p);
79          rotate(p);
80      }
81      pull(p);
82  }
```

```
83
84  node *access(node *p) {
85      node *j = nullptr;
86      for (node *i = p; i != nullptr; j = i, i = i->parent) {
87          splay(i);
88          i->val -= sum(j);
89          i->val += sum(i->child[1]);
90          i->child[1] = j;
91          pull(i);
92      }
93      splay(p);
94      return j;
95  }
96
97  void makeRoot(node *p) {
98      access(p);
99      reverse(p);
100 }
101
102 void link(node *p, node *q) {
103     makeRoot(p);
104     access(q);
105     p->parent = q;
106     q->val += sum(p);
107 }
108
109 void cut(node *p, node *q) {
110     makeRoot(p);
111     access(q);
112     p->parent = q->child[0] = nullptr;
113 }
114
115 node *pathMax(node *p, node *q) {
116     makeRoot(p);
117     access(q);
118     return max(q);
119 }
120
121 ll pathSize(node *p, node *q) {
122     makeRoot(p);
123     access(q);
124     return sz(q);
125 }
```

```
126
127  ll rootedSum(node *p) {
128      makeRoot(p);
129      return sum(p);
130  }
131
132  ll getSubtree(node *rt, node *v) {
133      makeRoot(rt);
134      access(v);
135      return v->val;
136  }
137
138  bool connected(node *p, node *q) {
139      access(p);
140      access(q);
141      return p->parent != nullptr;
142  }
143
144  void fix(node *p, ll v) {
145      access(p);
146      push(p);
147      // modify ...
148      p->val += v;
149      pull(p);
150  }
151
152  node *lca(node *z,node *x,node *y) {
153      makeRoot(z);
154      access(x);
155      return access(y);
156  }
157  } // namespace linkCutTree
158  using namespace linkCutTree;
```

2.15 lichao-tree.cpp

```
1  struct Line {
2    i64 k, b;
3    i64 operator()(i64 x) const { return k * x + b; }
4  };
5  template <i64 L, i64 R>
6  struct Segments {
7    struct Node {
8      optional<Line> s;
9      Node *l, *r;
10   };
11   Node *root;
12   Segments() : root(nullptr) {}
13   void add(i64 l, i64 r, i64 k, i64 b) {
14     auto rec = [&](auto &rec, Node *&p, i64 tl, i64 tr, Line s) -> void {
15       if (p == nullptr) p = new Node();
16       i64 tm = midpoint(tl, tr);
17       if (tl >= l and tr <= r) {
18         if (not p->s) return p->s = s, void();
19         auto t = p->s.value();
20         if (t(tl) >= s(tl)) {
21           if (t(tr) >= s(tr)) return;
22           if (t(tm) >= s(tm)) return rec(rec, p->r, tm + 1, tr, s);
23           return p->s = s, rec(rec, p->l, tl, tm, t);
24         }
25         if (t(tr) <= s(tr)) return p->s = s, void();
26         if (t(tm) <= s(tm)) return p->s = s, rec(rec, p->r, tm + 1, tr, t);
27         return rec(rec, p->l, tl, tm, s);
28       }
29       if (l <= tm) rec(rec, p->l, tl, tm, s);
30       if (r > tm) rec(rec, p->r, tm + 1, tr, s);
31     };
32     rec(rec, root, L, R, {k, b});
33   }
34   optional<i64> get(i64 x) {
35     optional<i64> res = {};
36     auto rec = [&](auto &rec, Node *p, i64 tl, i64 tr) -> void {
37       if (p == nullptr) return;
38       i64 tm = midpoint(tl, tr);
39       if (p->s) {
40         i64 y = p->s.value()(x);
41         if (not res or res.value() < y) res = y;
42       }
43       if (x <= tm)
44         rec(rec, p->l, tl, tm);
45       else
46         rec(rec, p->r, tm + 1, tr);
47     };
48     rec(rec, root, L, R);
49     return res;
50   }
```

```
51  };
```

2.16   Mo.cpp

```
1  /**
2   * #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
3   *      iota(all(s), 0);
4   *      sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
5   */
6
7  VI Mo(const vector<array<int, 3>> &Q) {
8      const int blk = 350;
9      vector<int> s(SZ(Q)), res = s;
10     iota(all(s), 0);
11     sort(all(s), [&](int i, int j) {
12         int u = Q[i][0] / blk, v = Q[j][0] / blk;
13         return u == v ? u % 2 ? Q[i][1] > Q[j][1] : Q[i][1] < Q[j][1] : u <
               v;
14     });
15     int L = 1, R = 0;
16     for (int qi : s) {
17         while (R < Q[qi][1]) R++, add(R);
18         while (L > Q[qi][0]) L--, add(L);
19         while (R > Q[qi][1]) del(R), R--;
20         while (L < Q[qi][0]) del(L), L++;
21         res[qi] = calc(Q[qi][2]);
22     }
23     return res;
24  }
```

2.17   moTree.cpp

```
1  /**
2   * Author: Simon Lindholm
3   * Date: 2019-12-28
4   * License: CC0
5   * Source: https://github.com/hoke-t/tamu-kactl/blob/master/content/data-
           structures/MoQueries.h
6   * Description: Answer interval or tree path queries by finding an
           approximate TSP through the queries,
7   * and moving from one query to the next by adding/removing points at the
           ends.
```

```
8   * If values are on tree edges, change \texttt{step} to add/remove the edge
           $(a, c)$ and remove the initial \texttt{add} call (but keep \texttt{in
           }).
9   * Time: O(N \sqrt Q)
10  * Status: stress-tested
11  */
12
13 void add(int ind, int end) { ... } // add a [ ind ] (end = 0 or 1)
14 void del(int ind, int end) { ... } // remove a [ ind ]
15 int calc() { ... } // compute current answer
16 vi mo(vector<pii> Q) {
17    int L = 0, R = 0, blk = 350; //  N/sqrt (Q)
18    vi s(sz(Q)), res = s;
19 #define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
20    iota(all(s), 0);
21    sort(all(s), [&](int s, int t) { return K(Q[s]) < K(Q[t]); });
22    for (int qi : s) {
23       pii q = Q[qi];
24       while (L > q.first) add(--L, 0);
25       while (R < q.second) add(R++, 1);
26       while (L < q.first) del(L++, 0);
27       while (R > q.second) del(--R, 1);
28       res[qi] = calc();
29    }
30    return res;
31 }
32 vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int root = 0) {
33    int N = sz(ed), pos[2] = {}, blk = 350; //  N/sqrt (Q)
34    vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N);
35    add(0, 0), in[0] = 1;
36    auto dfs = [&](int x, int p, int dep, auto & f) -> void {
37       par[x] = p;
38       L[x] = N;
39       if (dep) I[x] = N++;
40       for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
41       if (!dep) I[x] = N++;
42       R[x] = N;
43    };
44    dfs(root, -1, 0, dfs);
45 #define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] / blk & 1))
46    iota(all(s), 0);
47    sort(all(s), [&](int s, int t) { return K(Q[s]) < K(Q[t]); });
48    for (int qi : s) rep(end, 0, 2) {
```

```
49      int &a = pos[end], b = Q[qi][end], i = 0;
50  #define step(c) { if (in[c]) { del(a, end); in[a] = 0; } \
51  else { add(c, end); in[c] = 1; } a = c; }
52      while (!(L[b] <= L[a] && R[a] <= R[b]))
53        I[i++] = b, b = par[b];
54      while (a != b) step(par[a]);
55      while (i--) step(I[i]);
56      if (end) res[qi] = calc();
57    }
58    return res;
59  }
```

2.18   MSTMo.cpp

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   #define rep(i,a,n) for (int i=a;i<n;i++)
4   #define per(i,a,n) for (int i=n-1;i>=a;i--)
5   #define pb push_back
6   #define mp make_pair
7   #define all(x) (x).begin(),(x).end()
8   #define fi first
9   #define se second
10  #define SZ(x) ((int)(x).size())
11  typedef vector<int> VI;
12  typedef long long ll;
13  typedef pair<int,int> PII;
14  typedef double db;
15  mt19937 mrand(random_device{}());
16  const ll mod=1000000007;
17  int rnd(int x) { return mrand() % x;}
18  ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)
        res=res*a%mod;a=a*a%mod;}return res;}
19  ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}
20  // head
21
22  const int N=1010000;
23  int a[N];
24  namespace Mo {
25    int Q,l[N],r[N],f[N],l0,r0,ans[N],n;
26    VI ne[N];
27    struct point {
28      int x, y, o;
29      point(int a, int b, int c): x(a), y(b), o(c) {}
30    };
31    inline bool operator<(const point &a, const point &b) {
32      if (a.x != b.x) return a.x > b.x;
33      else return a.y < b.y;
34    }
35    vector<point> p;
36    struct edge {
37      int s, t, d;
38      edge(const point &a, const point &b): s(a.o), t(b.o),
39        d(abs(a.x - b.x) + abs(a.y - b.y)) {}
40    };
41    inline bool operator<(const edge &a, const edge &b) {return a.d < b.d;}
42    vector<edge> e;
43    int g[N],z[N];
44    int cc,cnt[101000];
45    void addedge() {
46      sort(all(p));
47      memset(g,0,sizeof(g));
48      z[0]=N;
49      rep(i,0,SZ(p)) z[i+1]=p[i].x-p[i].y;
50      rep(i,0,SZ(p)) {
51          int k = 0, t = p[i].x + p[i].y;
52          for (int j = t; j; j -= j & -j)
53              if (z[g[j]] < z[k]) k = g[j];
54          if (k) e.pb(edge(p[i], p[k - 1]));
55          k = z[i + 1];
56          for (int j = t; j <N; j += j & -j)
57              if (k < z[g[j]]) g[j] = i + 1;
58      }
59    }
60    void updata(int i, bool j,bool k=0) {
61      // j=1 insert  j=0 delete
62      // k=0 left k=1 right
63      if (j==1) {
64        cnt[a[i]]++;
65        if (cnt[a[i]]%2==0) cc++;
66      } else {
67        if (cnt[a[i]]%2==0) cc--;
68        cnt[a[i]]--;
69      }
70    }
71    void init(int l,int r) {
```

```
72      for (int i=l;i<=r;i++) {
73        cnt[a[i]]++;
74        if (cnt[a[i]]%2==0) cc++;
75      }
76    }
77    inline int query() {
78      return cc;
79    }
80    int find(int x) { if (f[x] != x) f[x] = find(f[x]); return f[x];}
81    void dfs(int i,int p) {
82      int l1 = l[i], r1 = r[i];
83      per(j,l1,l0) updata(j,1,0);
84      rep(j,r0+1,r1+1) updata(j,1,1);
85      rep(j,l0,l1) updata(j,0,0);
86      per(j,r1+1,r0+1) updata(j,0,1);
87      ans[i]=query();l0=l1;r0=r1;
88      rep(j,0,SZ(ne[i])) if (ne[i][j]!=p) dfs(ne[i][j],i);
89    }
90    void solve() {
91      p.clear();e.clear();
92      rep(i,1,Q+1) ans[i]=0;
93      rep(i,1,Q+1) p.pb(point(l[i],r[i],i));
94      addedge();
95      rep(i,0,SZ(p)) p[i].y =n-p[i].y+1;
96      addedge();
97      rep(i,0,SZ(p)) {
98        int j =n-p[i].x+1;
99        p[i].x = p[i].y; p[i].y = j;
100       }
101       addedge();
102       rep(i,0,SZ(p)) p[i].x=n-p[i].x+1;
103       addedge();
104       sort(all(e));
105       rep(i,1,Q+1) ne[i].clear(),f[i]=i;
106       rep(i,0,SZ(e)) {
107         int j=e[i].s,k=e[i].t;
108         if (find(j)!=find(k)) f[f[j]]=f[k],ne[j].pb(k),ne[k].pb(j);
109       }
110       l0=l[1];r0=r[1];
111       init(l0,r0);
112       dfs(1,0);
113     }
114   }
```

```
116   int main() {
117     scanf("%d",&Mo::n);
118     for (int i=1;i<=Mo::n;i++) scanf("%d",a+i);
119     scanf("%d",&Mo::Q);
120     rep(i,1,Mo::Q+1) scanf("%d%d",&Mo::l[i],&Mo::r[i]);
121     Mo::solve();
122     rep(i,1,Mo::Q+1) printf("%d\n",Mo::ans[i]);
123   }
```

2.19   psegt.cpp

```
1    struct node {
2      node *l, *r;
3      ll val, sz, add;
4    };
5
6    void pull(node *u) {
7      u->sz = 0, u->val = 0;
8      if (u->l) u->sz += u->l->sz, u->val += u->l->val;
9      if (u->r) u->sz += u->r->sz, u->val += u->r->val;
10   }
11
12   void push(node *u) {
13     if (u->add) {
14       if (u->l) {
15         node *p = new node();
16         *p = *u->l;
17         u->l = p;
18         p->add += u->add;
19         p->val += p->sz * u->add;
20       }
21       if (u->r) {
22         node *p = new node();
23         *p = *u->r;
24         u->r = p;
25         p->add += u->add;
26         p->val += p->sz * u->add;
27       }
28       u->add = 0;
29     }
30   }
31
```

```
32  node *build(int l, int r) {
33    node *p = new node();
34    p->add = 0;
35    if (l == r) {
36      p->l = p->r = nullptr;
37      p->val = a[l];
38      p->sz = 1;
39    } else {
40      int mid = (l + r) >> 1;
41      p->l = build(l, mid);
42      p->r = build(mid + 1, r);
43      pull(p);
44    }
45    return p;
46  }
47
48  ll query(node *v, int l, int r, int ql, int qr) {
49    if (ql == l && qr == r) {
50      return v->val;
51    } else {
52      push(v);
53      int mid = (l + r) >> 1;
54      if (qr <= mid)
55        return query(v->l, l, mid, ql, qr);
56      else if (ql > mid)
57        return query(v->r, mid + 1, r, ql, qr);
58      else
59        return query(v->l, l, mid, ql, mid) +
60               query(v->r, mid + 1, r, mid + 1, qr);
61    }
62  }
63
64  node *modify(node *v, int l, int r, int ql, int qr, ll x) {
65    if (ql == l && qr == r) {
66      node *p = new node();
67      *p = *v;
68      p->add += x;
69      p->val += p->sz * x;
70      return p;
71    } else {
72      push(v);
73      int mid = (l + r) >> 1;
74      node *p = new node();
```

```
75      *p = *v;
76      if (qr <= mid)
77        p->l = modify(v->l, l, mid, ql, qr, x);
78      else if (ql > mid)
79        p->r = modify(v->r, mid + 1, r, ql, qr, x);
80      else
81        p->l = modify(v->l, l, mid, ql, mid, x),
82        p->r = modify(v->r, mid + 1, r, mid + 1, qr, x);
83      pull(p);
84      return p;
85    }
86  }
```

2.20   rollbackMo.cpp

```
1   VI rollbackMo(const vector<array<int, 3>> &Q) {
2       const int blk = 350;
3       vector<VI> s(SZ(Q));
4       vector<int> BF, res(SZ(Q));
5       for (int i = 0; i < SZ(Q); i++) {
6           int u = Q[i][0] / blk, v = Q[i][1] / blk;
7           if (u == v) BF.push_back(i);
8           else s[u].push_back(i);
9       }
10      for (int i = 0; i < SZ(Q); i++)
11          sort(all(s[i]), [&](int i, int j) { return Q[i][1] < Q[j][1]; });
12      for (int qi : BF) {
13          for (int i = Q[qi][0]; i <= Q[qi][1]; i++)
14              add(i);
15          res[qi] = calc(Q[qi][2]);
16          for (int i = Q[qi][0]; i <= Q[qi][1]; i++)
17              del(i);
18      }
19      for (const auto &v : s) {
20          if (v.empty()) continue;
21          int next_blk = (Q[v.back()][0] / blk + 1) * blk;
22          int L = next_blk, R = next_blk - 1;
23          for (int qi : v) {
24              while (R < Q[qi][1]) R++, add(R);
25              while (L > Q[qi][0]) L--, add(L);
26              res[qi] = calc(Q[qi][2]);
27              while (L < next_blk) del(L), L++;
28          }
```

```
29        for (int i = next_blk; i <= R; i++)
30            del(i);
31    }
32    return res;
33 }
```

2.21   segtree.cpp

```
1  struct info {
2      ll sum;
3      int sz;
4      friend info operator+(const info &a, const info &b) {
5          return {(a.sum + b.sum) % mod, a.sz + b.sz};
6      }
7  };
8
9  struct tag {
10     ll add, mul;
11     friend tag operator+(const tag &a, const tag &b) {
12         tag res = {(a.add * b.mul + b.add) % mod, a.mul * b.mul % mod};
13         return res;
14     }
15 };
16 info operator+(const info &a, const tag &b) {
17     return {(a.sum * b.mul + a.sz * b.add) % mod, a.sz};
18 }
19
20 struct node {
21     info val;
22     tag t;
23 } seg[maxn << 2];
24
25 void update(int id) {
26     seg[id].val = seg[id * 2].val + seg[id * 2 + 1].val;
27 }
28 void settag(int id, tag t) {
29     seg[id].val = seg[id].val + t;
30     seg[id].t = seg[id].t + t;
31 }
32 void pushdown(int id) {
33     if (seg[id].t.mul == 1 and seg[id].t.add == 0) return;
34     settag(id * 2, seg[id].t);
35     settag(id * 2 + 1, seg[id].t);
```

```
36        seg[id].t.mul = 1;
37        seg[id].t.add = 0;
38 }
39 void build(int l, int r, int id) {
40     seg[id].t = {0, 1};
41     if (l == r) {
42         seg[id].val = {a[l], 1};
43     } else {
44         int mid = (l + r) >> 1;
45         build(l, mid, id * 2);
46         build(mid + 1, r, id * 2 + 1);
47         update(id);
48     }
49 }
50 void change(int l, int r, int id, int ql, int qr, tag t) {
51     if (l == ql && r == qr) {
52         settag(id, t);
53     } else {
54         int mid = (l + r) >> 1;
55         pushdown(id);
56         if (qr <= mid) {
57             change(l, mid, id * 2, ql, qr, t);
58         } else if (ql > mid) {
59             change(mid + 1, r, id * 2 + 1, ql, qr, t);
60         } else {
61             change(l, mid, id * 2, ql, mid, t);
62             change(mid + 1, r, id * 2 + 1, mid + 1, qr, t);
63         }
64         update(id);
65     }
66 }
67 info query(int l, int r, int id, int ql, int qr) {
68     if (l == ql && r == qr) {
69         return seg[id].val;
70     } else {
71         int mid = (l + r) >> 1;
72         pushdown(id);
73         if (qr <= mid)
74             return query(l, mid, id * 2, ql, qr);
75         else if (ql > mid)
76             return query(mid + 1, r, id * 2 + 1, ql, qr);
77         else
78             return query(l, mid, id * 2, ql, mid) +
```

```
79              query(mid + 1, r, id * 2 + 1, mid + 1, qr);
80      }
81  }
82  ll search(int l, int r, int id, int ql, int qr, int d) {
83      if (ql == l && qr == r) {
84          int mid = (l + r) / 2;
85          // if (l != r) pushdown(id); ...
86          if (seg[id].val < d)
87              return -1;
88          else {
89              if (l == r)
90                  return l;
91              else if (seg[id * 2].val >= d)
92                  return search(l, mid, id * 2, ql, mid, d);
93              else
94                  return search(mid + 1, r, id * 2 + 1, mid + 1, qr, d);
95          }
96      } else {
97          int mid = (l + r) >> 1;
98          // pushdown(id); ...
99          if (qr <= mid)
100             return search(l, mid, id * 2, ql, qr, d);
101         else if (ql > mid)
102             return search(mid + 1, r, id * 2 + 1, ql, qr, d);
103         else {
104             int tmp = search(l, mid, id * 2, ql, mid, d);
105             if (tmp != -1)
106                 return tmp;
107             else
108                 return search(mid + 1, r, id * 2 + 1, mid + 1, qr, d);
109         }
110     }
111 }
```

2.22   segtreefast.cpp

```
1  /**
2   * Author: Lucian Bicsi
3   * Description: Very fast and quick segment tree.
4   * Only useful for easy invariants. 0-indexed.
5   * Range queries are half-open.
6   */
7  #pragma once
```

```
8
9  struct SegmTree {
10    vector<int> T; int n;
11    SegmTree(int n) : T(2 * n, (int)2e9), n(n) {}
12
13    void Update(int pos, int val) {
14      for (T[pos += n] = val; pos > 1; pos /= 2)
15        T[pos / 2] = min(T[pos], T[pos ^ 1]);
16    }
17
18    int Query(int b, int e) {
19      int res = (int)2e9;
20      for (b += n, e += n; b < e; b /= 2, e /= 2) {
21        if (b % 2) res = min(res, T[b++]);
22        if (e % 2) res = min(res, T[--e]);
23      }
24      return res;
25    }
26  };
```

2.23   SparseTable.cpp

```
1  template <typename T, class F = function<T(const T&, const T&)>>
2  class SparseTable {
3  public:
4    int n;
5    vector<vector<T>> mat;
6    F func;
7
8    SparseTable(const vector<T>& a, const F& f) : func(f) {
9      n = static_cast<int>(a.size());
10     int max_log = 32 - __builtin_clz(n);
11     mat.resize(max_log);
12     mat[0] = a;
13     for (int j = 1; j < max_log; j++) {
14       mat[j].resize(n - (1 << j) + 1);
15       for (int i = 0; i <= n - (1 << j); i++) {
16         mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
17       }
18     }
19   }
20
21   T get(int from, int to) const {
```

```
22        assert(0 <= from && from <= to && to <= n - 1);
23        int lg = 32 - __builtin_clz(to - from + 1) - 1;
24        return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
25    }
26  };
```

2.24   SparseTable2D.cpp

```
1   // lg[1] = 0;
2   // rep(i, 2, N - 1) {
3   //     lg[i] = lg[i / 2] + 1;
4   // }
5   // int k = log2(r - l + 1); very slow!!!
6   // int k = __lg(r - l + 1);
7   // int k = lg[r - l + 1];
8   // int k = 32 - __builtin_clz(r - l + 1) - 1;
9   vector<vector<int>> sparse[12];
10
11  int query(int x, int y, int d) {
12    int k = __lg(d);
13    int s = d - bit(k);
14    return min({sparse[k][x][y], sparse[k][x + s][y], sparse[k][x][y + s],
          sparse[k][x + s][y + s]});
15  }
16
17  void build() {
18    rep(i, 1, n) rep(j, 1, m) sparse[0][i][j] = mat[i][j];
19    rep(k, 1, 11) rep(i, 1, n) rep(j, 1, m) {
20      int d = bit(k - 1);
21      if (i + d > n || j + d > m) continue;
22      sparse[k][i][j] = min({sparse[k - 1][i][j], sparse[k - 1][i + d][j],
          sparse[k - 1][i][j + d], sparse[k - 1][i + d][j + d]});
23    }
24  }
```

2.25   treap.cpp

```
1   /**
2    *    author:  tourist
3    *    created: 07.10.2022 20:32:03
4   **/
5   #include <bits/stdc++.h>
6
7   using namespace std;
8
9   #ifdef LOCAL
10  #include "algo/debug.h"
11  #else
12  #define debug(...) 42
13  #endif
14
15  mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
16
17  class node {
18  public:
19    int id;
20    node* l;
21    node* r;
22    node* p;
23    bool rev;
24    int sz;
25    // declare extra variables:
26    long long P;
27    long long add;
28    long long x;
29
30    node(int _id, long long _x) {
31      id = _id;
32      l = r = p = nullptr;
33      rev = false;
34      sz = 1;
35      // init extra variables:
36      P = rng();
37      add = 0;
38      x = _x;
39    }
40
41    // push everything else:
42    void push_stuff() {
43      if (add != 0) {
44        if (l != nullptr) {
45          l->unsafe_apply(add);
46        }
47        if (r != nullptr) {
48          r->unsafe_apply(add);
49        }
```

```
50        add = 0;
51      }
52    }
53
54    void unsafe_reverse() {
55      push_stuff();
56      rev ^= 1;
57      swap(l, r);
58      pull();
59    }
60
61    // apply changes:
62    void unsafe_apply(long long delta) {
63      add += delta;
64      x += delta;
65    }
66
67    void push() {
68      if (rev) {
69        if (l != nullptr) {
70          l->unsafe_reverse();
71        }
72        if (r != nullptr) {
73          r->unsafe_reverse();
74        }
75        rev = 0;
76      }
77      push_stuff();
78    }
79
80    void pull() {
81      sz = 1;
82      if (l != nullptr) {
83        l->p = this;
84        sz += l->sz;
85      }
86      if (r != nullptr) {
87        r->p = this;
88        sz += r->sz;
89      }
90    }
91  };
92
```

```
93   void debug_node(node* v, string pref = "") {
94   #ifdef LOCAL
95     if (v != nullptr) {
96       debug_node(v->r, pref + "␣");
97       cerr << pref << "-" << "␣" << v->id << '\n';
98       debug_node(v->l, pref + "␣");
99     } else {
100      cerr << pref << "-" << "␣" << "nullptr" << '\n';
101    }
102  #endif
103  }
104
105  namespace treap {
106
107  pair<node*, int> find(node* v, const function<int(node*)> &go_to) {
108    // go_to returns: 0 -- found; -1 -- go left; 1 -- go right
109    // find returns the last vertex on the descent and its go_to
110    if (v == nullptr) {
111      return {nullptr, 0};
112    }
113    int dir;
114    while (true) {
115      v->push();
116      dir = go_to(v);
117      if (dir == 0) {
118        break;
119      }
120      node* u = (dir == -1 ? v->l : v->r);
121      if (u == nullptr) {
122        break;
123      }
124      v = u;
125    }
126    return {v, dir};
127  }
128
129  node* get_leftmost(node* v) {
130    return find(v, [&](node*) { return -1; }).first;
131  }
132
133  node* get_rightmost(node* v) {
134    return find(v, [&](node*) { return 1; }).first;
135  }
```

24

```
136
137  node* get_kth(node* v, int k) { // 0-indexed
138    pair<node*, int> p = find(v, [&](node * u) {
139      if (u->l != nullptr) {
140        if (u->l->sz > k) {
141          return -1;
142        }
143        k -= u->l->sz;
144      }
145      if (k == 0) {
146        return 0;
147      }
148      k--;
149      return 1;
150    });
151    return (p.second == 0 ? p.first : nullptr);
152  }
153
154  int get_pos(node* v) { // 0-indexed
155    int k = (v->l != nullptr ? v->l->sz : 0);
156    while (v->p != nullptr) {
157      if (v == v->p->r) {
158        k++;
159        if (v->p->l != nullptr) {
160          k += v->p->l->sz;
161        }
162      }
163      v = v->p;
164    }
165    return k;
166  }
167
168  node* get_root(node* v) {
169    while (v->p != nullptr) {
170      v = v->p;
171    }
172    return v;
173  }
174
175  pair<node*, node*> split(node* v, const function<bool(node*)> &is_right) {
176    if (v == nullptr) {
177      return {nullptr, nullptr};
178    }
```

```
179    v->push();
180    if (is_right(v)) {
181      pair<node*, node*> p = split(v->l, is_right);
182      if (p.first != nullptr) {
183        p.first->p = nullptr;
184      }
185      v->l = p.second;
186      v->pull();
187      return {p.first, v};
188    } else {
189      pair<node*, node*> p = split(v->r, is_right);
190      v->r = p.first;
191      if (p.second != nullptr) {
192        p.second->p = nullptr;
193      }
194      v->pull();
195      return {v, p.second};
196    }
197  }
198
199  pair<node*, node*> split_cnt(node* v, int k) {
200    if (v == nullptr) {
201      return {nullptr, nullptr};
202    }
203    v->push();
204    int left_and_me = (v->l != nullptr ? v->l->sz : 0) + 1;
205    if (k < left_and_me) {
206      pair<node*, node*> p = split_cnt(v->l, k);
207      if (p.first != nullptr) {
208        p.first->p = nullptr;
209      }
210      v->l = p.second;
211      v->pull();
212      return {p.first, v};
213    } else {
214      pair<node*, node*> p = split_cnt(v->r, k - left_and_me);
215      v->r = p.first;
216      if (p.second != nullptr) {
217        p.second->p = nullptr;
218      }
219      v->pull();
220      return {v, p.second};
221    }
```

```
222   }
223
224   node* merge(node* v, node* u) {
225     if (v == nullptr) {
226       return u;
227     }
228     if (u == nullptr) {
229       return v;
230     }
231     if (v->P > u->P) {
232 //      if (rng() % (v->sz + u->sz) < (unsigned int) v->sz) {
233       v->push();
234       v->r = merge(v->r, u);
235       v->pull();
236       return v;
237     } else {
238       u->push();
239       u->l = merge(v, u->l);
240       u->pull();
241       return u;
242     }
243   }
244
245   int count_left(node* v, const function<bool(node*)> &is_right) {
246     if (v == nullptr) {
247       return 0;
248     }
249     v->push();
250     if (is_right(v)) {
251       return count_left(v->l, is_right);
252     }
253     return (v->l != nullptr ? v->l->sz : 0) + 1 + count_left(v->r, is_right);
254   }
255
256   int count_less(node* v, long long val) {
257     int res = 0;
258     while (v != nullptr) {
259       v->push();
260       if (v->x >= val) {
261         v = v->l;
262       } else {
263         res += (v->l != nullptr ? v->l->sz : 0) + 1;
264         v = v->r;
```

```
265       }
266     }
267     return res;
268   }
269
270   node* add(node* r, node* v, const function<bool(node*)> &go_left) {
271     pair<node*, node*> p = split(r, go_left);
272     return merge(p.first, merge(v, p.second));
273   }
274
275   node* remove(node* v) { // returns the new root
276     v->push();
277     node* x = v->l;
278     node* y = v->r;
279     node* p = v->p;
280     v->l = v->r = v->p = nullptr;
281     v->push();
282     v->pull(); // now v might be reusable...
283     node* z = merge(x, y);
284     if (p == nullptr) {
285       if (z != nullptr) {
286         z->p = nullptr;
287       }
288       return z;
289     }
290     if (p->l == v) {
291       p->l = z;
292     }
293     if (p->r == v) {
294       p->r = z;
295     }
296     while (true) {
297       p->push();
298       p->pull();
299       if (p->p == nullptr) {
300         break;
301       }
302       p = p->p;
303     }
304     return p;
305   }
306
307   node* next(node* v) {
```

```
308    if (v->r == nullptr) {
309      while (v->p != nullptr && v->p->r == v) {
310        v = v->p;
311      }
312      return v->p;
313    }
314    v->push();
315    v = v->r;
316    while (v->l != nullptr) {
317      v->push();
318      v = v->l;
319    }
320    return v;
321  }
322
323  node* prev(node* v) {
324    if (v->l == nullptr) {
325      while (v->p != nullptr && v->p->l == v) {
326        v = v->p;
327      }
328      return v->p;
329    }
330    v->push();
331    v = v->l;
332    while (v->r != nullptr) {
333      v->push();
334      v = v->r;
335    }
336    return v;
337  }
338
339  int get_size(node* v) {
340    return (v != nullptr ? v->sz : 0);
341  }
342
343  template<typename... T>
344  void Apply(node* v, T... args) {
345    v->unsafe_apply(args...);
346  }
347
348  void reverse(node* v) {
349    v->unsafe_reverse();
350  }
```

```
351
352  // extra of mine
353  long long lower(node* u, long long x) {
354    if (u == nullptr)
355      return numeric_limits<long long>::min();
356    else if (x <= u->x)
357      return lower(u->l, x);
358    else
359      return max(u->x, lower(u->r, x));
360  }
361
362  long long upper(node* u, long long x) {
363    if (u == nullptr)
364      return numeric_limits<long long>::max();
365    else if (u->x <= x)
366      return upper(u->r, x);
367    else
368      return min(u->x, upper(u->l, x));
369  }
370
371  }  // namespace treap
372
373  using namespace treap;
374
375  int n;
376
377  int main() {
378    ios::sync_with_stdio(false);
379    cin.tie(0);
380    node* root = nullptr;
381    cin >> n;
382    for (int i = 1; i <= n; i++) {
383      int op;
384      long long x;
385      cin >> op >> x;
386      switch (op) {
387        case 1: {
388          root = add(root, new node(x, x), [&](node * u) {
389            return x < u->x;
390          });
391          break;
392        }
393        case 2: {
```

27

```
394        auto [pt, w] = find(root, [&](node * u) {
395          if (x < u->x) return -1;
396          else if (x == u->x) return 0;
397          else return 1;
398        });
399        assert(w == 0);
400        root = remove(pt);
401        break;
402      }
403      case 3: {
404        cout << count_less(root, x) + 1 << '\n';
405        break;
406      }
407      case 4: {
408        cout << get_kth(root, x - 1)->x << '\n';
409        break;
410      }
411      case 5: {
412        cout << lower(root, x) << '\n';
413        break;
414      }
415      case 6: {
416        cout << upper(root, x) << '\n';
417        break;
418      }
419    }
420  }
421 }
```

2.26   UnionFindRollback.cpp

```
1  vector<int> fa(n);
2  iota(all(fa), 0);
3  vector<int> sz(n, 1);
4  vector<pair<int, int>> ops;
5
6  auto Get = [&](int i) {
7      while (i != fa[i]) {
8          i = fa[i];
9      }
10     return i;
11 };
12 auto Unite = [&](int i, int j) {
13     i = Get(i), j = Get(j);
14     if (i == j) {
15         return;
16     }
17     if (sz[i] > sz[j]) {
18         swap(i, j);
19     }
20     ops.emplace_back(i, fa[i]);
21     fa[i] = j;
22     ops.emplace_back(~j, sz[j]);
23     sz[j] += sz[i];
24 };
25 auto RollBack = [&](int T) {
26     while (SZ(ops) > T) {
27         auto [i, j] = ops.back();
28         ops.pop_back();
29         if (i >= 0) {
30             fa[i] = j;
31         } else {
32             sz[~i] = j;
33         }
34     }
35 };
36
37 ll ans = 0;
38 auto Dfs = [&](auto &&Dfs, int l, int r) -> void {
39     if (l == r) {
40         for (auto [x, y] : g[l]) {
41             x = Get(x);
42             y = Get(y);
43             ans += 1ll * sz[x] * sz[y];
44         }
45     } else {
46         int mid = midpoint(l, r);
47         {
48             int save = SZ(ops);
49             for (int i = mid + 1; i <= r; i++) {
50                 for (auto [x, y] : g[i]) {
51                     Unite(x, y);
52                 }
53             }
54             Dfs(Dfs, l, mid);
55             RollBack(save);
```

```
56              }
57          {
58                  int save = SZ(ops);
59                  for (int i = l; i <= mid; i++) {
60                      for (auto [x, y] : g[i]) {
61                          Unite(x, y);
62                      }
63                  }
64                  Dfs(Dfs, mid + 1, r);
65                  RollBack(save);
66              }
67          }
68  };
69  Dfs(Dfs, 0, n - 1);
```

2.27 树哈希.cpp

```
1   basic_string<int> e[maxn];
2   ull hashv[maxn];
3   ull seed1, seed2, seed3, seed4;
4
5   ull f(ull x) { return x * x * x * seed1 + x * seed2; }
6   ull h(ull x) { return f(x) ^ ((x & seed3) >> 31) ^ ((x & seed4) << 31); }
7
8   void dfs1(int u, int fa) {
9       hashv[u] = 1;
10      for (auto v : e[u]) if (v != fa) {
11              dfs1(v, u);
12              hashv[u] += h(hashv[v]);
13          }
14  }
15
16  void dfs2(int u, int fa, ull fv) {
17  // for each root
18      hashv[u] += fv;
19      for (auto v : e[u]) if (v != fa) {
20              dfs2(v, u, h(hashv[u] - h(hashv[v])));
21          }
22  }
23
24  void solve() {
25      seed1 = rng(), seed2 = rng();
26      seed3 = rng(), seed4 = rng();
```

```
27      cin >> n;
28      rep(i, 2, n) {
29          int u, v;
30          cin >> u >> v;
31          e[u].pb(v);
32          e[v].pb(u);
33      }
34      dfs1(1, 0);
35      sort(hashv + 1, hashv + n + 1);
36      n = unique(hashv + 1, hashv + n + 1) - hashv - 1;
37      cout << n << '\n';
38  }
```

2.28 树链剖分 segtree.cpp

```
1   int n, m, a[N];
2   vector<int> e[N];
3   int l[N], r[N], idx[N];
4   int sz[N], hs[N], tot, top[N], dep[N], fa[N];
5
6   struct info {
7       int maxv, sum;
8   };
9
10  info operator + (const info &l, const info &r) {
11      return (info){max(l.maxv, r.maxv), l.sum + r.sum};
12  }
13
14  struct node {
15      info val;
16  } seg[N * 4];
17
18  // [l, r]
19
20  void update(int id) {
21      seg[id].val = seg[id * 2].val + seg[id * 2 + 1].val;
22  }
23
24  void build(int id, int l, int r) {
25      if (l == r) {
26          // l号点， DFS序中第l个点
27          seg[id].val = {a[idx[l]], a[idx[l]]};
28      } else {
```

```
29            int mid = (l + r) / 2;
30            build(id * 2, l, mid);
31            build(id * 2 + 1, mid + 1, r);
32            update(id);
33        }
34    }
35
36    void change(int id, int l, int r, int pos, int val) {
37        if (l == r) {
38            seg[id].val = {val, val};
39        } else {
40            int mid = (l + r) / 2;
41            if (pos <= mid) change(id * 2, l, mid, pos, val);
42            else change(id * 2 + 1, mid + 1, r, pos, val);
43            update(id);
44        }
45    }
46
47    info query(int id, int l, int r, int ql, int qr) {
48        if (l == ql && r == qr) return seg[id].val;
49        int mid = (l + r) / 2;
50        if (qr <= mid) return query(id * 2, l, mid, ql, qr);
51        else if (ql > mid) return query(id * 2 + 1, mid + 1, r, ql,qr);
52        else {
53            return query(id * 2, l, mid, ql, mid) +
54                    query(id * 2 + 1, mid + 1, r, mid + 1, qr);
55        }
56    }
57
58    // 第一遍 DFS，子树大小，重儿子，父亲，深度
59    void dfs1(int u,int f) {
60        sz[u] = 1;
61        hs[u] = -1;
62        fa[u] = f;
63        dep[u] = dep[f] + 1;
64        for (auto v : e[u]) {
65            if (v == f) continue;
66            dfs1(v, u);
67            sz[u] += sz[v];
68            if (hs[u] == -1 || sz[v] > sz[hs[u]])
69                hs[u] = v;
70        }
71    }
```

```
72
73    // 第二遍 DFS，每个点 DFS 序，重链上的链头的元素。
74    void dfs2(int u, int t) {
75        top[u] = t;
76        l[u] = ++tot;
77        idx[tot] = u;
78        if (hs[u] != -1) {
79            dfs2(hs[u], t);
80        }
81        for (auto v : e[u]) {
82            if (v != fa[u] && v != hs[u]) {
83                dfs2(v, v);
84            }
85        }
86        r[u] = tot;
87    }
88
89    int LCA(int u, int v) {
90        while (top[u] != top[v]) {
91            if (dep[top[u]] < dep[top[v]]) v = fa[top[v]];
92            else u = fa[top[u]];
93        }
94        if (dep[u] < dep[v]) return u;
95        else return v;
96    }
97
98    info query(int u,int v) {
99        info ans{(int)-1e9, 0};
100        while (top[u] != top[v]) {
101            if (dep[top[u]] < dep[top[v]]) {
102                ans = ans + query(1, 1, n, l[top[v]], l[v]);
103                v = fa[top[v]];
104            } else {
105                ans = ans + query(1, 1, n, l[top[u]], l[u]);
106                u = fa[top[u]];
107            }
108        }
109        if (dep[u] <= dep[v]) ans = ans + query(1, 1, n, l[u], l[v]);
110        else ans = ans + query(1, 1, n, l[v], l[u]);
111        return ans;
112    }
```

2.29 笛卡尔树.cpp

```cpp
int a[maxn], l[maxn], r[maxn], root;
int ans[maxn], tot;

void build() {
    stack<int> stk;
    for (int i = 1; i <= n; i++) {
        int last = 0;
        while (!stk.empty() && a[stk.top()] > a[i]) {
            last = stk.top();
            stk.pop();
        }
        if (stk.empty())
            root = i;
        else
            r[stk.top()] = i;
        l[i] = last;
        stk.push(i);
    }
}

void dfs(int c, int L, int R) {
    ans[c] = ++tot;
    if (l[c]) dfs(l[c], L, c - 1);
    if (r[c]) dfs(r[c], c + 1, R);
}
```

2.30 线段树合并.cpp

```cpp
struct node {
  int sz, sum;
  node *l, *r;
  node() : sz(0), sum(0), l(nullptr), r(nullptr) {}
} pool[N * 20], *cur = pool;

node *newnode() {
  return cur++;
}

void upd(node *rt) {
  if (not rt) return;
  rt->sum = rt->sz > 0;
  if (rt->l) rt->sum += rt->l->sum;
```

```cpp
  if (rt->r) rt->sum += rt->r->sum;
}

node *modify(node *rt, int l, int r, int pos, int d) {
  if (not rt) rt = newnode();
  if (l == r) {
    rt->sz += d;
    upd(rt);
    return rt;
  } else {
    int md = (l + r) >> 1;
    if (pos <= md)
      rt->l = modify(rt->l, l, md, pos, d);
    else
      rt->r = modify(rt->r, md + 1, r, pos, d);
    upd(rt);
    return rt;
  }
}

node *merge(node *u, node *v, int l, int r) {
  if (not u) return v;
  if (not v) return u;
  if (l == r) {
    u->sz += v->sz;
    upd(u);
    return u;
  } else {
    int md = (l + r) >> 1;
    u->l = merge(u->l, v->l, l, md);
    u->r = merge(u->r, v->r, md + 1, r);
    upd(u);
    return u;
  }
}

ll query(node *rt, int l, int r) {
  if (not rt) return 0;
  return rt->sum;
}

pair<node *, node *> split(node *rt, int l, int r, int ql, int qr) {
  if (not rt) return {nullptr, nullptr};
```

```
58      if (ql == l && qr == r) {
59        return {nullptr, rt};
60      } else {
61        int md = (l + r) >> 1;
62        if (qr <= md) {
63          auto [p1, p2] = split(rt->l, l, md, ql, qr);
64          rt->l = p1;
65          upd(rt);
66          if (not p2) return {rt, nullptr};
67          node *u = newnode();
68          u->l = p2;
69          upd(u);
70          return {rt, u};
71        } else if (ql > md) {
72          auto [p1, p2] = split(rt->r, md + 1, r, ql, qr);
73          rt->r = p1;
74          upd(rt);
75          if (not p2) return {rt, nullptr};
76          node *u = newnode();
77          u->r = p2;
78          upd(u);
79          return {rt, u};
80        } else {
81          auto [p1, p2] = split(rt->l, l, md, ql, md);
82          auto [p3, p4] = split(rt->r, md + 1, r, md + 1, qr);
83          rt->l = p1, rt->r = p3;
84          upd(rt);
85          if (not p2 and not p4) return {rt, nullptr};
86          node *u = newnode();
87          u->l = p2, u->r = p4;
88          upd(u);
89          return {rt, u};
90        }
91      }
92    }
```

3 DP

3.1 Convex hull optimization.cpp

```
1  array<ll, 3> a[maxn];
2  int q[maxn];
```

```
3   ll ans[maxn];
4
5   ll X(int p) {
6       return 2ll * a[p][0];
7   }
8   ll Y(int p) {
9       return a[p][0] * a[p][0] + a[p][1];
10  }
11  ldb slope(int x, int y) {
12      return (ldb)(Y(y) - Y(x)) / (X(y) - X(x));
13  }
14  void solve() {
15      cin >> n;
16      int head = 1, rear = 0;
17      rep(i, 1, n) {
18          cin >> a[i][0] >> a[i][1];
19          a[i][2] = i;
20      }
21      sort(a + 1, a + n + 1);
22
23      rep(i, 1, n) {
24          while (head < rear && slope(q[rear], i) <= slope(q[rear], q[rear -
                  1])) rear--;
25          q[++rear] = i;
26      }
27      rep(i, 1, n) {
28          ll k = -a[i][0];
29          while (head < rear && slope(q[head], q[head + 1]) <= k) head++;
30          ans[a[i][2]] = (a[i][0] + a[q[head]][0]) * (a[i][0] + a[q[head]][0])
                  + a[i][1] + a[q[head]][1];
31      }
32      rep(i, 1, n) cout << ans[i] << '\n';
33  }
```

3.2 DivideAndConquerDP.cpp

```
1  ll w[N][N],sum[N][N],opt[N],dp[805][N];
2
3  ll calc(int i,int j) { return sum[j][j]-sum[j][i]-sum[i][j]+sum[i][i]; }
4
5  void rec(int d,int l,int r,int optl,int optr) {
6      if (l>r) return;
7      int md=(l+r)>>1;
```

```
8      rep(i,optl,optr) if (dp[d-1][i]+calc(i,md)<dp[d][md]) {
9          dp[d][md]=dp[d-1][i]+calc(i,md);
10         opt[md]=i;
11     }
12     rec(d,l,md-1,optl,opt[md]);
13     rec(d,md+1,r,opt[md],optr);
14 }
```

3.3 有依赖决策单调.cpp

```
1  pair<int, int> stk[N];
2  auto calc = [&](int i, int j) { ... } // dp[j] -> dp[i]
3  int h = 0, t = 0;
4  stk[t++] = {1, 0}; // {left, opt}
5
6  for (int i = 1; i <= n; i++) {
7      if (h < t && stk[h].first < i) stk[h].first++;
8      if (h + 1 < t && stk[h].first >= stk[h + 1].first) ++h;
9      dp[i] = calc(i, stk[h].second);
10     while (h < t && calc(stk[t - 1].first, stk[t - 1].second) >= calc(stk[t
           - 1].first, i))
11         --t;
12     if (h < t) {
13         int l = stk[t - 1].first, r = n + 1;
14         while (l + 1 < r) {
15             int md = (l + r) >> 1;
16             if (calc(md, stk[t - 1].second) < calc(md, i)) l = md; else r =
                   md;
17         }
18         if (r <= n) stk[t++] = {r, i};
19     } else stk[t++] = {i, i};
20 }
```

4 Geometry

4.1 1 (1).cpp

```
1  typedef double db;
2  const db EPS = 1e-9;
3
4  inline int sign(db a) { return a < -EPS ? -1 : a > EPS; }
5
6  inline int cmp(db a, db b) { return sign(a - b); }
7
8  struct P {
9      db x, y;
10     P() {}
11     P(db _x, db _y) : x(_x), y(_y) {}
12     P operator+(P p) { return {x + p.x, y + p.y}; }
13     P operator-(P p) { return {x - p.x, y - p.y}; }
14     P operator*(db d) { return {x * d, y * d}; }
15     P operator/(db d) { return {x / d, y / d}; }
16
17     bool operator<(P p) const {
18         int c = cmp(x, p.x);
19         if (c) return c == -1;
20         return cmp(y, p.y) == -1;
21     }
22
23     bool operator==(P o) const {
24         return cmp(x, o.x) == 0 && cmp(y, o.y) == 0;
25     }
26
27     db dot(P p) { return x * p.x + y * p.y; }
28     db det(P p) { return x * p.y - y * p.x; }
29
30     db distTo(P p) { return (*this - p).abs(); }
31     db alpha() { return atan2(y, x); }
32     void read() { cin >> x >> y; }
33     void write() {cout << "(" << x << "," << y << ")" << endl;}
34     db abs() { return sqrt(abs2());}
35     db abs2() { return x * x + y * y; }
36     P rot90() { return P(-y, x);}
37     P unit() { return *this / abs(); }
38     int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0)
           ; }
39     P rot(db an) { return {x * cos(an) - y * sin(an), x * sin(an) + y * cos(
           an)}; }
40 };
41
42 #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
43 #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
44
45 // 直线 p1p2, q1q2 是否恰有一个交点
46 bool chkLL(P p1, P p2, P q1, P q2) {
```

```cpp
47        db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
48        return sign(a1 + a2) != 0;
49  }
50
51  // 求直线 p1p2, q1q2 的交点
52  P isLL(P p1, P p2, P q1, P q2) {
53        db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
54        return (p1 * a2 + p2 * a1) / (a1 + a2);
55  }
56
57  // 判断区间 [l1, r1], [l2, r2] 是否相交
58  bool intersect(db l1, db r1, db l2, db r2) {
59        if (l1 > r1) swap(l1, r1); if (l2 > r2) swap(l2, r2);
60        return !( cmp(r1, l2) == -1 || cmp(r2, l1) == -1 );
61  }
62
63  // 线段 p1p2, q1q2 相交
64  bool isSS(P p1, P p2, P q1, P q2) {
65        return intersect(p1.x, p2.x, q1.x, q2.x) && intersect(p1.y, p2.y, q1.y,
              q2.y) &&
66                crossOp(p1, p2, q1) * crossOp(p1, p2, q2) <= 0 && crossOp(q1, q2,
                    p1)
67                * crossOp(q1, q2, p2) <= 0;
68  }
69
70  // 线段 p1p2, q1q2 严格相交
71  bool isSS_strict(P p1, P p2, P q1, P q2) {
72        return crossOp(p1, p2, q1) * crossOp(p1, p2, q2) < 0 && crossOp(q1, q2,
              p1)
73                * crossOp(q1, q2, p2) < 0;
74  }
75
76  // m 在 a 和 b 之间
77  bool isMiddle(db a, db m, db b) {
78        /*if (a > b) swap(a, b);
79        return cmp(a, m) <= 0 && cmp(m, b) <= 0;*/
80        return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
81  }
82
83  bool isMiddle(P a, P m, P b) {
84        return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
85  }
86
87  // 点 p 在线段 p1p2 上
88  bool onSeg(P p1, P p2, P q) {
89        return crossOp(p1, p2, q) == 0 && isMiddle(p1, q, p2);
90  }
91  // q1q2 和 p1p2 的交点 在 p1p2 上?
92
93  // 点 p 严格在 p1p2 上
94  bool onSeg_strict(P p1, P p2, P q) {
95        return crossOp(p1, p2, q) == 0 && sign((q - p1).dot(p1 - p2)) * sign((q
              - p2).dot(p1 - p2)) < 0;
96  }
97
98  // 求 q 到 直线p1p2 的投影 (垂足)    : p1 != p2
99  P proj(P p1, P p2, P q) {
100       P dir = p2 - p1;
101       return p1 + dir * (dir.dot(q - p1) / dir.abs2());
102  }
103
104  // 求 q 以 直线p1p2 为轴的反射
105  P reflect(P p1, P p2, P q) {
106       return proj(p1, p2, q) * 2 - q;
107  }
108
109  // 求 q 到 线段p1p2 的最小距离
110  db nearest(P p1, P p2, P q) {
111       if (p1 == p2) return p1.distTo(q);
112       P h = proj(p1, p2, q);
113       if (isMiddle(p1, h, p2))
114            return q.distTo(h);
115       return min(p1.distTo(q), p2.distTo(q));
116  }
117
118  // 求 线段p1p2 与 线段q1q2 的距离
119  db disSS(P p1, P p2, P q1, P q2) {
120       if (isSS(p1, p2, q1, q2)) return 0;
121       return min(min(nearest(p1, p2, q1), nearest(p1, p2, q2)), min(nearest(q1
              , q2, p1), nearest(q1, q2, p2)));
122  }
123
124  // 极角排序
125  sort(p, p + n, [&](P a, P b) {
126       int qa = a.quad(), qb = b.quad();
127       if (qa != qb) return qa < qb;
```

```
128         else return sign(a.det(b)) > 0;
129 });
```

4.2   1 (2).cpp

```
 1  db area(vector<P> ps){
 2      db ret = 0; rep(i,0,ps.size()) ret += ps[i].det(ps[(i+1)%ps.size()]);
 3      return ret/2;
 4  }
 5
 6  int contain(vector<P> ps, P p){ //2:inside,1:on_seg,0:outside
 7      int n = ps.size(), ret = 0;
 8      rep(i,0,n){
 9          P u=ps[i],v=ps[(i+1)%n];
10          if(onSeg(u,v,p)) return 1;
11          if(cmp(u.y,v.y)<=0) swap(u,v);
12          if(cmp(p.y,u.y) > 0 || cmp(p.y,v.y) <= 0) continue;
13          ret ^= crossOp(p,u,v) > 0;
14      }
15      return ret*2;
16  }
17
18  vector<P> convexHull(vector<P> ps) {
19      int n = ps.size(); if(n <= 1) return ps;
20      sort(ps.begin(), ps.end());
21      vector<P> qs(n * 2); int k = 0;
22      for (int i = 0; i < n; qs[k++] = ps[i++])
23          while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
24      for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
25          while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
26      qs.resize(k - 1);
27      return qs;
28  }
29
30  vector<P> convexHullNonStrict(vector<P> ps) {
31      //caution: need to unique the Ps first
32      int n = ps.size(); if(n <= 1) return ps;
33      sort(ps.begin(), ps.end());
34      vector<P> qs(n * 2); int k = 0;
35      for (int i = 0; i < n; qs[k++] = ps[i++])
36          while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
37      for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
38          while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
```

```
39      qs.resize(k - 1);
40      return qs;
41  }
42
43  db convexDiameter(vector<P> ps){
44      int n = ps.size(); if(n <= 1) return 0;
45      int is = 0, js = 0; rep(k,1,n) is = ps[k]<ps[is]?k:is, js = ps[js] < ps[
            k]?k:js;
46      int i = is, j = js;
47      db ret = ps[i].distTo(ps[j]);
48      do{
49          if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j]) >= 0)
50              (++j)%=n;
51          else
52              (++i)%=n;
53          ret = max(ret,ps[i].distTo(ps[j]));
54      }while(i!=is || j!=js);
55      return ret;
56  }
57
58  vector<P> convexCut(const vector<P>&ps, P q1, P q2) {
59      vector<P> qs;
60      int n = ps.size();
61      rep(i,0,n){
62          P p1 = ps[i], p2 = ps[(i+1)%n];
63          int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
64          if(d1 >= 0) qs.push_back(p1);
65          if(d1 * d2 < 0) qs.push_back(isLL(p1,p2,q1,q2));
66      }
67      return qs;
68  }
69
70  void reorderPolygon(vector<P> &ps) {
71      size_t pos = 0;
72      for(size_t i = 1; i < ps.size(); i++){
73          if(ps[i].y < ps[pos].y || (ps[i].y == ps[pos].y && ps[i].x < ps[pos
                ].x))
74              pos = i;
75      }
76      rotate(ps.begin(), ps.begin() + pos, ps.end());
77  }
78
79  vector<P> minkowski(vector<P> p, vector<P> q){
```

```
80        if(p.empty()) return q;
81        // the first vertex must be the lowest
82        reorderPolygon(p);
83        reorderPolygon(q);
84        // must ensure cyclic indexing
85        p.push_back(p[0]);
86        p.push_back(p[1]);
87        q.push_back(q[0]);
88        q.push_back(q[1]);
89        // main part
90        vector<P> result;
91        size_t i = 0, j = 0;
92        while(i < p.size() - 2 || j < q.size() - 2){
93            result.push_back(p[i] + q[j]);
94            auto cross = (p[i + 1] - p[i]).det(q[j + 1] - q[j]);
95            if(cross >= 0 && i < SZ(p) - 2)
96                ++i;
97            if(cross <= 0 && j < SZ(q) - 2)
98                ++j;
99        }
100       return result;
101   }
102
103   bool convexContain(const vector<P> &l, P p, bool strict = true) {
104       int a = 1, b = l.size() - 1, r = !strict;
105       if (l.size() < 3) return r && onSeg(l[0], l.back(), p);
106       if (crossOp(l[0], l[a], l[b]) > 0) swap(a, b);
107       if (crossOp(l[0], l[a], p) >= r || crossOp(l[0], l[b], p)<= -r)
108           return false;
109       while (abs(a - b) > 1) {
110           int c = (a + b) / 2;
111           (crossOp(l[0], l[c], p) > 0 ? b : a) = c;
112       }
113       return sign(cross(l[a], l[b], p)) < r;
114   }
```

4.3   1 (3).cpp

```
1    int type(P o1,db r1,P o2,db r2){
2        db d = o1.distTo(o2);
3        if(cmp(d,r1+r2) == 1) return 4;
4        if(cmp(d,r1+r2) == 0) return 3;
5        if(cmp(d,abs(r1-r2)) == 1) return 2;
```

```
6        if(cmp(d,abs(r1-r2)) == 0) return 1;
7        return 0;
8    }
9
10   vector<P> isCL(P o,db r,P p1,P p2){
11       if (cmp(abs((o-p1).det(p2-p1)/p1.distTo(p2)),r)>0) return {};
12       db x = (p1-o).dot(p2-p1), y = (p2-p1).abs2(), d = x * x - y * ((p1-o).
             abs2() - r*r);
13       d = max(d,(db)0.0); P m = p1 - (p2-p1)*(x/y), dr = (p2-p1)*(sqrt(d)/y);
14       return {m-dr,m+dr}; //along dir: p1->p2
15   }
16
17   vector<P> isCC(P o1, db r1, P o2, db r2) { //need to check whether two
          circles are the same
18       db d = o1.distTo(o2);
19       if (cmp(d, r1 + r2) == 1) return {};
20       if (cmp(d,abs(r1-r2))==-1) return {};
21       d = min(d, r1 + r2);
22       db y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y * y);
23       P dr = (o2 - o1).unit();
24       P q1 = o1 + dr * y, q2 = dr.rot90() * x;
25       return {q1-q2,q1+q2};//along circle 1
26   }
27
28   // extanCC, intanCC : -r2, tanCP : r2 = 0
29   vector<pair<P, P>> tanCC(P o1, db r1, P o2, db r2) {
30       P d = o2 - o1;
31       db dr = r1 - r2, d2 = d.abs2(), h2 = d2 - dr * dr;
32       if (sign(d2) == 0|| sign(h2) < 0) return {};
33       h2 = max((db)0.0, h2);
34       vector<pair<P, P>> ret;
35       for (db sign : {-1, 1}) {
36           P v = (d * dr + d.rot90() * sqrt(h2) * sign) / d2;
37           ret.push_back({o1 + v * r1, o2 + v * r2});
38       }
39       if (sign(h2) == 0) ret.pop_back();
40       return ret;
41   }
42
43   db rad(P p1,P p2){
44       return atan2l(p1.det(p2),p1.dot(p2));
45   }
46
```

```
47  db areaCT(db r, P p1, P p2){
48      vector<P> is = isCL(P(0,0),r,p1,p2);
49      if(is.empty()) return r*r*rad(p1,p2)/2;
50      bool b1 = cmp(p1.abs2(),r*r) == 1, b2 = cmp(p2.abs2(), r*r) == 1;
51      if(b1 && b2){
52          P md=(is[0]+is[1])/2;
53          if(sign((p1-md).dot(p2-md)) <= 0)
54              return r*r*(rad(p1,is[0]) + rad(is[1],p2))/2 + is[0].det(is[1])
                      /2;
55          else return r*r*rad(p1,p2)/2;
56      }
57      if(b1) return (r*r*rad(p1,is[0]) + is[0].det(p2))/2;
58      if(b2) return (p1.det(is[1]) + r*r*rad(is[1],p2))/2;
59      return p1.det(p2)/2;
60  }
61
62  P inCenter(P A, P B, P C) {
63      double a = (B - C).abs(), b = (C - A).abs(), c = (A - B).abs();
64      return (A * a + B * b + C * c) / (a + b + c);
65  }
66
67  P circumCenter(P a, P b, P c) {
68      P bb = b - a, cc = c - a;
69      double db = bb.abs2(), dc = cc.abs2(), d = 2 * bb.det(cc);
70      return a - P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
71  }
72
73  P othroCenter(P a, P b, P c) {
74      P ba = b - a, ca = c - a, bc = b - c;
75      double Y = ba.y * ca.y * bc.y,
76      A = ca.x * ba.y - ba.x * ca.y,
77      x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
78      y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
79      return {x0, y0};
80  }
81
82  pair<P,db> min_circle(vector<P> ps){
83      random_shuffle(ps.begin(), ps.end());
84      int n = ps.size();
85      P o = ps[0]; db r = 0;
86      rep(i,1,n) if(o.distTo(ps[i]) > r + EPS){
87          o = ps[i], r = 0;
88          rep(j,0,i) if(o.distTo(ps[j]) > r + EPS){
89              o = (ps[i] + ps[j]) / 2; r = o.distTo(ps[i]);
90              rep(k,0,j) if(o.distTo(ps[k]) > r + EPS){
91                  o = circumCenter(ps[i],ps[j],ps[k]);
92                  r = o.distTo(ps[i]);
93              }
94          }
95      }
96      return {o,r};
97  }
```

4.4   all.cpp

```
1  typedef double db;
2  const db EPS = 1e-9;
3
4  inline int sign(db a) { return a < -EPS ? -1 : a > EPS; }
5
6  inline int cmp(db a, db b){ return sign(a-b); }
7
8  struct P {
9      db x, y;
10     P() {}
11     P(db _x, db _y) : x(_x), y(_y) {}
12     P operator+(P p) { return {x + p.x, y + p.y}; }
13     P operator-(P p) { return {x - p.x, y - p.y}; }
14     P operator*(db d) { return {x * d, y * d}; }
15     P operator/(db d) { return {x / d, y / d}; }
16
17     bool operator<(P p) const {
18         int c = cmp(x, p.x);
19         if (c) return c == -1;
20         return cmp(y, p.y) == -1;
21     }
22
23     bool operator==(P o) const{
24         return cmp(x,o.x) == 0 && cmp(y,o.y) == 0;
25     }
26
27     db dot(P p) { return x * p.x + y * p.y; }
28     db det(P p) { return x * p.y - y * p.x; }
29
30     db distTo(P p) { return (*this-p).abs(); }
31     db alpha() { return atan2(y, x); }
```

```
32        void read() { cin>>x>>y; }
33        void write() {cout<<"("<<x<<","<<y<<")"<<endl;}
34        db abs() { return sqrt(abs2());}
35        db abs2() { return x * x + y * y; }
36        P rot90() { return P(-y,x);}
37        P unit() { return *this/abs(); }
38        int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0)
              ; }
39        P rot(db an){ return {x*cos(an)-y*sin(an),x*sin(an) + y*cos(an)}; }
40   };
41
42   struct L{ //ps[0] -> ps[1]
43       P ps[2];
44       P& operator[](int i) { return ps[i]; }
45       P dir() { return ps[1] - ps[0]; }
46       bool include(P p) { return sign((ps[1] - ps[0]).det(p - ps[0])) > 0; }
47       L push(){ // push eps outward
48           const double eps = 1e-6;
49           P delta = (ps[1] - ps[0]).rot90().unit() * eps;
50           return {{ps[0] - delta, ps[1] - delta}};
51       }
52   };
53
54   #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
55   #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
56
57   bool chkLL(P p1, P p2, P q1, P q2) {
58       db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
59       return sign(a1+a2) != 0;
60   }
61
62   P isLL(P p1, P p2, P q1, P q2) {
63       db a1 = cross(q1, q2, p1), a2 = -cross(q1, q2, p2);
64       return (p1 * a2 + p2 * a1) / (a1 + a2);
65   }
66
67   P isLL(L l1,L l2){ return isLL(l1[0],l1[1],l2[0],l2[1]); }
68
69   bool intersect(db l1,db r1,db l2,db r2){
70       if(l1>r1) swap(l1,r1); if(l2>r2) swap(l2,r2);
71       return !( cmp(r1,l2) == -1 || cmp(r2,l1) == -1 );
72   }
73
74   bool isSS(P p1, P p2, P q1, P q2){
75       return intersect(p1.x,p2.x,q1.x,q2.x) && intersect(p1.y,p2.y,q1.y,q2.y)
             &&
76       crossOp(p1,p2,q1) * crossOp(p1,p2,q2) <= 0 && crossOp(q1,q2,p1)
77               * crossOp(q1,q2,p2) <= 0;
78   }
79
80   bool isSS_strict(P p1, P p2, P q1, P q2){
81       return crossOp(p1,p2,q1) * crossOp(p1,p2,q2) < 0 && crossOp(q1,q2,p1)
82               * crossOp(q1,q2,p2) < 0;
83   }
84
85   bool isMiddle(db a, db m, db b) {
86       return sign(a - m) == 0 || sign(b - m) == 0 || (a < m != b < m);
87   }
88
89   bool isMiddle(P a, P m, P b) {
90       return isMiddle(a.x, m.x, b.x) && isMiddle(a.y, m.y, b.y);
91   }
92
93   bool onSeg(P p1, P p2, P q){
94       return crossOp(p1,p2,q) == 0 && isMiddle(p1, q, p2);
95   }
96
97   bool onSeg_strict(P p1, P p2, P q){
98       return crossOp(p1,p2,q) == 0 && sign((q-p1).dot(p1-p2)) * sign((q-p2).
             dot(p1-p2)) < 0;
99   }
100
101  P proj(P p1, P p2, P q) {
102      P dir = p2 - p1;
103      return p1 + dir * (dir.dot(q - p1) / dir.abs2());
104  }
105
106  P reflect(P p1, P p2, P q){
107      return proj(p1,p2,q) * 2 - q;
108  }
109
110  db nearest(P p1,P p2,P q){
111      P h = proj(p1,p2,q);
112      if(isMiddle(p1,h,p2))
113          return q.distTo(h);
114      return min(p1.distTo(q),p2.distTo(q));
```

```cpp
115    }
116
117    db disSS(P p1, P p2, P q1, P q2){
118        if(isSS(p1,p2,q1,q2)) return 0;
119        return min(min(nearest(p1,p2,q1),nearest(p1,p2,q2)), min(nearest(q1,q2,
               p1),nearest(q1,q2,p2)));
120    }
121
122    db rad(P p1,P p2){
123        return atan2l(p1.det(p2),p1.dot(p2));
124    }
125
126    db incircle(P p1, P p2, P p3){
127        db A = p1.distTo(p2);
128        db B = p2.distTo(p3);
129        db C = p3.distTo(p1);
130        return sqrtl(A*B*C/(A+B+C));
131    }
132
133    //polygon
134
135    db area(vector<P> ps){
136        db ret = 0; rep(i,0,ps.size()) ret += ps[i].det(ps[(i+1)%ps.size()]);
137        return ret/2;
138    }
139
140    int contain(vector<P> ps, P p){ //2:inside,1:on_seg,0:outside
141        int n = ps.size(), ret = 0;
142        rep(i,0,n){
143            P u=ps[i],v=ps[(i+1)%n];
144            if(onSeg(u,v,p)) return 1;
145            if(cmp(u.y,v.y)<=0) swap(u,v);
146            if(cmp(p.y,u.y) >0 || cmp(p.y,v.y) <= 0) continue;
147            ret ^= crossOp(p,u,v) > 0;
148        }
149        return ret*2;
150    }
151
152    vector<P> convexHull(vector<P> ps) {
153        int n = ps.size(); if(n <= 1) return ps;
154        sort(ps.begin(), ps.end());
155        vector<P> qs(n * 2); int k = 0;
156        for (int i = 0; i < n; qs[k++] = ps[i++])
157            while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
158        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
159            while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) <= 0) --k;
160        qs.resize(k - 1);
161        return qs;
162    }
163
164    vector<P> convexHullNonStrict(vector<P> ps) {
165        //caution: need to unique the Ps first
166        int n = ps.size(); if(n <= 1) return ps;
167        sort(ps.begin(), ps.end());
168        vector<P> qs(n * 2); int k = 0;
169        for (int i = 0; i < n; qs[k++] = ps[i++])
170            while (k > 1 && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
171        for (int i = n - 2, t = k; i >= 0; qs[k++] = ps[i--])
172            while (k > t && crossOp(qs[k - 2], qs[k - 1], ps[i]) < 0) --k;
173        qs.resize(k - 1);
174        return qs;
175    }
176
177    db convexDiameter(vector<P> ps){
178        int n = ps.size(); if(n <= 1) return 0;
179        int is = 0, js = 0; rep(k,1,n) is = ps[k]<ps[is]?k:is, js = ps[js] < ps[
               k]?k:js;
180        int i = is, j = js;
181        db ret = ps[i].distTo(ps[j]);
182        do{
183            if((ps[(i+1)%n]-ps[i]).det(ps[(j+1)%n]-ps[j]) >= 0)
184                (++j)%=n;
185            else
186                (++i)%=n;
187            ret = max(ret,ps[i].distTo(ps[j]));
188        }while(i!=is || j!=js);
189        return ret;
190    }
191
192    vector<P> convexCut(const vector<P>&ps, P q1, P q2) {
193        vector<P> qs;
194        int n = ps.size();
195        rep(i,0,n){
196            P p1 = ps[i], p2 = ps[(i+1)%n];
197            int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
198            if(d1 >= 0) qs.pb(p1);
```

```
199          if(d1 * d2 < 0) qs.pb(isLL(p1,p2,q1,q2));
200      }
201      return qs;
202 }
203
204 //min_dist
205
206 db min_dist(vector<P>&ps,int l,int r){
207      if(r-l<=5){
208          db ret = 1e100;
209          rep(i,l,r) rep(j,l,i) ret = min(ret,ps[i].distTo(ps[j]));
210          return ret;
211      }
212      int m = (l+r)>>1;
213      db ret = min(min_dist(ps,l,m),min_dist(ps,m,r));
214      vector<P> qs; rep(i,l,r) if(abs(ps[i].x-ps[m].x)<= ret) qs.pb(ps[i]);
215      sort(qs.begin(), qs.end(),[](P a,P b) -> bool {return a.y<b.y; });
216      rep(i,1,qs.size()) for(int j=i-1;j>=0&&qs[j].y>=qs[i].y-ret;--j)
217          ret = min(ret,qs[i].distTo(qs[j]));
218      return ret;
219 }
220
221 int type(P o1,db r1,P o2,db r2){
222      db d = o1.distTo(o2);
223      if(cmp(d,r1+r2) == 1) return 4;
224      if(cmp(d,r1+r2) == 0) return 3;
225      if(cmp(d,abs(r1-r2)) == 1) return 2;
226      if(cmp(d,abs(r1-r2)) == 0) return 1;
227      return 0;
228 }
229
230 vector<P> isCL(P o,db r,P p1,P p2){
231      db x = (p1-o).dot(p2-p1), y = (p2-p1).abs2(), d = x * x - y * ((p1-o).
             abs2() - r*r);
232      if(sign(d) < 0) return {};
233      d = max(d,0.0); P m = p1 - (p2-p1)*(x/y), dr = (p2-p1)*(sqrt(d)/y);
234      return {m-dr,m+dr}; //along dir: p1->p2
235 }
236
237 vector<P> isCC(P o1, db r1, P o2, db r2) { //need to check whether two
         circles are the same
238      db d = o1.distTo(o2);
239      if (cmp(d, r1 + r2) == 1) return {};
```

```
240      d = min(d, r1 + r2);
241      db y = (r1 * r1 + d * d - r2 * r2) / (2 * d), x = sqrt(r1 * r1 - y * y);
242      P dr = (o2 - o1).unit();
243      P q1 = o1 + dr * y, q2 = dr.rot90() * x;
244      return {q1-q2,q1+q2};//along circle 1
245 }
246
247 vector<P> tanCP(P o, db r, P p) {
248      db x = (p - o).abs2(), d = x - r * r;
249      if (sign(d) <= 0) return {}; // on circle => no tangent
250      P q1 = o + (p - o) * (r * r / x);
251      P q2 = (p - o).rot90() * (r * sqrt(d) / x);
252      return {q1-q2,q1+q2}; //counter clock-wise
253 }
254
255
256 vector<L> extanCC(P o1, db r1, P o2, db r2) {
257      vector<L> ret;
258      if (cmp(r1, r2) == 0) {
259          P dr = (o2 - o1).unit().rot90() * r1;
260          ret.pb({{o1 + dr, o2 + dr}}), ret.pb({{o1 - dr, o2 - dr}});
261      } else {
262          P p = (o2 * r1 - o1 * r2) / (r1 - r2);
263          vector<P> ps = tanCP(o1, r1, p), qs = tanCP(o2, r2, p);
264          rep(i,0,min(ps.size(),qs.size())) ret.pb({{ps[i], qs[i]}}); //c1
                 counter-clock wise
265      }
266      return ret;
267 }
268
269 vector<L> intanCC(P o1, db r1, P o2, db r2) {
270      vector<L> ret;
271      P p = (o1 * r2 + o2 * r1) / (r1 + r2);
272      vector<P> ps = tanCP(o1,r1,p), qs = tanCP(o2,r2,p);
273      rep(i,0,min(ps.size(),qs.size())) ret.pb({{ps[i], qs[i]}}); //c1 counter
             -clock wise
274      return ret;
275 }
276
277 db areaCT(db r, P p1, P p2){
278      vector<P> is = isCL(P(0,0),r,p1,p2);
279      if(is.empty()) return r*r*rad(p1,p2)/2;
280      bool b1 = cmp(p1.abs2(),r*r) == 1, b2 = cmp(p2.abs2(), r*r) == 1;
```

```
281        if(b1 && b2){
282            if(sign((p1-is[0]).dot(p2-is[0])) <= 0 &&
283                sign((p1-is[0]).dot(p2-is[0])) <= 0)
284            return r*r*(rad(p1,is[0]) + rad(is[1],p2))/2 + is[0].det(is[1])/2;
285            else return r*r*rad(p1,p2)/2;
286        }
287        if(b1) return (r*r*rad(p1,is[0]) + is[0].det(p2))/2;
288        if(b2) return (p1.det(is[1]) + r*r*rad(is[1],p2))/2;
289        return p1.det(p2)/2;
290    }
291
292    bool parallel(L l0, L l1) { return sign( l0.dir().det( l1.dir() ) ) == 0; }
293
294    bool sameDir(L l0, L l1) { return parallel(l0, l1) && sign(l0.dir().dot(l1.
        dir()) ) == 1; }
295
296    bool cmp (P a,  P b) {
297        if (a.quad() != b.quad()) {
298            return a.quad() < b.quad();
299        } else {
300            return sign( a.det(b) ) > 0;
301        }
302    }
303
304    bool operator < (L l0, L l1) {
305        if (sameDir(l0, l1)) {
306            return l1.include(l0[0]);
307        } else {
308            return cmp( l0.dir(), l1.dir() );
309        }
310    }
311
312    bool check(L u, L v, L w) {
313        return w.include(isLL(u,v));
314    }
315
316    vector<P> halfPlaneIS(vector<L> &l) {
317        sort(l.begin(), l.end());
318        deque<L> q;
319        for (int i = 0; i < (int)l.size(); ++i) {
320            if (i && sameDir(l[i], l[i - 1])) continue;
321            while (q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i
                ])) q.pop_back();
322            while (q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
323            q.push_back(l[i]);
324        }
325        while (q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q
            .pop_back();
326        while (q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front
            ();
327        vector<P> ret;
328        for (int i = 0; i < (int)q.size(); ++i) ret.push_back(isLL(q[i], q[(i +
            1) % q.size()]));
329        return ret;
330    }
331
332    P inCenter(P A, P B, P C) {
333        double a = (B - C).abs(), b = (C - A).abs(), c = (A - B).abs();
334        return (A * a + B * b + C * c) / (a + b + c);
335    }
336
337    P circumCenter(P a, P b, P c) {
338        P bb = b - a, cc = c - a;
339        double db = bb.abs2(), dc = cc.abs2(), d = 2 * bb.det(cc);
340        return a - P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc) / d;
341    }
342
343    P othroCenter(P a, P b, P c) {
344        P ba = b - a, ca = c - a, bc = b - c;
345        double Y = ba.y * ca.y * bc.y,
346        A = ca.x * ba.y - ba.x * ca.y,
347        x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
348        y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
349        return {x0, y0};
350    }
```

4.5 圆面积并.cpp

```
1    db intergal(db x,db y,db r,db L,db R){
2        return r*r*(R-L) + x*r*(sinl(R) - sinl(L)) + y*r*(-cosl(R) + cosl(L));
3    }
4
5    db calc_area_circle(P c,db r,db L,db R){
6        return intergal(c.x,c.y,r,L,R) / 2;
7    }
8
```

```
 9  db norm(db x){
10      while(x < 0) x += 2 * PI;
11      while(x > 2 * PI) x -= 2 * PI;
12      return x;
13  }
14
15  P cs[N]; db rs[N];
16
17  void work(){
18      vector<int> cand = {};
19      rep(i,0,m){
20          bool ok = 1;
21          rep(j,0,m) if(i!=j){
22              if(rs[j] > rs[i] + EPS && rs[i] + cs[i].distTo(cs[j]) <= rs[j] +
                     EPS){
23                  ok = 0; break;
24              }
25              if(cs[i] == cs[j] && cmp(rs[i],rs[j]) == 0 && j < i){
26                  ok = 0; break;
27              }
28          }
29          if(ok) cand.pb(i);
30      }
31
32      rep(i,0,cand.size()) cs[i] = cs[cand[i]], rs[i] = rs[cand[i]];
33      m = cand.size();
34
35      db area = 0;
36
37      //work
38      rep(i,0,m){
39          vector<pair<db,int>> ev = {{0,0},{2*PI,0}};
40
41          int cur = 0;
42
43          rep(j,0,m) if(j!=i){
44              auto ret = isCC(cs[i],rs[i],cs[j],rs[j]);
45              if(!ret.empty()){
46                  db l = (ret[0] - cs[i]).alpha();
47                  db r = (ret[1] - cs[i]).alpha();
48                  l = norm(l); r = norm(r);
49                  ev.pb({l,1});ev.pb({r,-1});
50                  if(l > r) ++cur;
```

```
51              }
52          }
53
54          sort(ev.begin(), ev.end());
55          rep(j,0,ev.size() - 1){
56              cur += ev[j].se;
57              if(cur == 0){
58                  area += calc_area_circle(cs[i],rs[i],ev[j].fi,ev[j+1].fi);
59              }
60          }
61      }
62  }
```

5 Graph

5.1 bellmanford.cpp

```
 1  vector<PII> e[N];
 2
 3  template <typename T>
 4  void add(int u, int v, T w) {
 5      e[u].eb(v, w);
 6  }
 7
 8  template <typename T>
 9  vector<T> bellmanford(vector<pair<int, T>> *g, int start) {
10      // assert(0 <= start && start < g.n);
11      // maybe use inf = numeric_limits<T>::max() / 4
12      const T inf = numeric_limits<T>::max() / 4;
13      vector<T> dist(n, inf);
14      dist[start] = 0;
15      int cnt = 0;
16      while (true) {
17          bool upd = 0;
18          cnt++;
19          for (int i = 0; i < n; i++) {
20              for (auto [to, cost] : e[i]) {
21                  if (dist[to] > dist[i] + cost) {
22                      upd = 1;
23                      dist[to] = dist[i] + cost;
24                  }
25              }
```

```
26              }
27              if (!upd || cnt == n) {
28                  break;
29              }
30          }
31      return dist;
32      // returns inf if there's no path
33  }
```

5.2 BlockCutTree.cpp

```
1   struct BlockCutTree {
2       int n;
3       std::vector<std::vector<int>> adj;
4       std::vector<int> dfn, low, stk;
5       int cnt, cur;
6       std::vector<std::pair<int, int>> edges;
7
8       BlockCutTree() {}
9       BlockCutTree(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          stk.clear();
19          cnt = cur = 0;
20          edges.clear();
21      }
22
23      void addEdge(int u, int v) {
24          adj[u].push_back(v);
25          adj[v].push_back(u);
26      }
27
28      void dfs(int x) {
29          stk.push_back(x);
30          dfn[x] = low[x] = cur++;
31
32          for (auto y : adj[x]) {
```

```
33              if (dfn[y] == -1) {
34                  dfs(y);
35                  low[x] = std::min(low[x], low[y]);
36                  if (low[y] == dfn[x]) {
37                      int v;
38                      do {
39                          v = stk.back();
40                          stk.pop_back();
41                          edges.emplace_back(n + cnt, v);
42                      } while (v != y);
43                      edges.emplace_back(x, n + cnt);
44                      cnt++;
45                  }
46              } else {
47                  low[x] = std::min(low[x], dfn[y]);
48              }
49          }
50      }
51
52      std::pair<int, std::vector<std::pair<int, int>>> work() {
53          for (int i = 0; i < n; i++) {
54              if (dfn[i] == -1) {
55                  stk.clear();
56                  dfs(i);
57              }
58          }
59          return {cnt, edges};
60      }
61  };
```

5.3 boruvka.cpp

```
1   /**
2    * while component > 1:
3    *     for each component:
4    *         find select[i]
5    *     for each component:
6    *         if select[i] != i:
7    *             merge(i, select[i])
8    *             component--
9    */
10
11  ll ans = 0, cnt = n;
```

```cpp
12    while (cnt > 1) {
13        fill(select + 1, select + n + 1, -1);
14        vector<int> cand;
15        for (int i = 1; i <= n; i++) {
16            cand.push_back(col[i]);
17        }
18        ranges::sort(cand);
19        cand.erase(unique(all(cand)), cand.end());
20
21        for (auto id : cand) {
22            for (auto x : S[id]) remove(x);
23            for (auto x : S[id]) {
24                auto [opt, w] = get(x);
25                if (select[id] == -1 || w < mn[id]) {
26                    select[id] = opt, mn[id] = w;
27                }
28            }
29            for (auto x : S[id]) insert(x);
30        }
31
32        for (int i = 1; i <= n; i++) if (col[i] == i) {
33            int j = col[select[i]];
34            if (i == j) continue;
35            ans += mn[i];
36            merge(i, j);
37            cnt--;
38        }
39 }
```

5.4   dijfast.cpp

```cpp
1  vector<PII> e[N];
2
3  template <typename T>
4  void add(int u, int v, T w) {
5      e[u].eb(v, w);
6  }
7
8  template <typename T>
9  vector<T> dijkstra(vector<pair<int, T>> *g, int start) {
10     // assert(0 <= start && start < g.n);
11     // maybe use inf = numeric_limits<T>::max() / 4
12     vector<T> dist(n, numeric_limits<T>::max());
```

```cpp
13        priority_queue<pair<T, int>, vector<pair<T, int>>, greater<pair<T, int
            >>> s;
14        dist[start] = 0;
15        s.emplace(dist[start], start);
16        while (!s.empty()) {
17            T expected = s.top().first;
18            int i = s.top().second;
19            s.pop();
20            if (dist[i] != expected) {
21                continue;
22            }
23            for (auto [to, cost] : g[i]) {
24                if (dist[i] + cost < dist[to]) {
25                    dist[to] = dist[i] + cost;
26                    s.emplace(dist[to], to);
27                }
28            }
29        }
30        return dist;
31        // returns numeric_limits<T>::max() if there's no path
32 }
```

5.5   dijkstra.cpp

```cpp
1  vector<PII> e[N];
2
3  template <typename T>
4  void add(int u, int v, T w) {
5      e[u].eb(v, w);
6  }
7
8  template <typename T>
9  vector<T> dijkstra(vector<pair<int, T>> *g, int start) {
10     // assert(0 <= start && start < g.n);
11     // maybe use inf = numeric_limits<T>::max() / 4
12     const T inf = numeric_limits<T>::max();
13     vector<T> dist(n, inf);
14     vector<int> was(n, 0);
15     dist[start] = 0;
16     while (true) {
17         int cur = -1;
18         for (int i = 0; i < n; i++) {
19             if (was[i] || dist[i] == inf) continue;
```

```
20          if (cur == -1 || dist[i] < dist[cur]) {
21              cur = i;
22          }
23       }
24       if (cur == -1 || dist[cur] == inf) {
25           break;
26       }
27       was[cur] = 1;
28       for (auto [to, cost] : g[cur]) {
29           dist[to] = min(dist[to], dist[cur] + cost);
30       }
31   }
32   return dist;
33   // returns inf if there's no path
34 }
```

5.6   dinic.cpp

```
1  template<typename T>
2  struct FlowGraph {
3      static const int V = 1015;
4      static const int E = 100015;
5      int s, t, vtot;
6      int head[V], etot;
7      int dis[V], cur[V];
8      struct edge {
9          int v, nxt;
10         T f;
11     } e[E * 2];
12     void addedge(int u, int v, T f) {
13         e[etot] = {v, head[u], f};
14         head[u] = etot++;
15         e[etot] = {u, head[v], 0};
16         head[v] = etot++;
17     }
18     bool bfs() {
19         for (int i = 1; i <= vtot; i++) {
20             dis[i] = 0;
21             cur[i] = head[i];
22         }
23         queue<int> q;
24         q.push(s); dis[s] = 1;
25         while (!q.empty()) {
26             int u = q.front(); q.pop();
27             for (int i = head[u]; i != -1; i = e[i].nxt) {
28                 if (e[i].f && !dis[e[i].v]) {
29                     int v = e[i].v;
30                     dis[v] = dis[u] + 1;
31                     if (v == t) return true;
32                     q.push(v);
33                 }
34             }
35         }
36         return false;
37     }
38     T dfs(int u, T m) {
39         if (u == t) return m;
40         T flow = 0;
41         for (int i = cur[u]; i != -1; cur[u] = i = e[i].nxt) {
42             if (e[i].f && dis[e[i].v] == dis[u] + 1) {
43                 T f = dfs(e[i].v, min(m, e[i].f));
44                 e[i].f -= f;
45                 e[i ^ 1].f += f;
46                 m -= f;
47                 flow += f;
48                 if (!m) break;
49             }
50         }
51         if (!flow) dis[u] = -1;
52         return flow;
53     }
54     T dinic() {
55         T flow = 0;
56         while (bfs()) flow += dfs(s, numeric_limits<T>::max());
57         return flow;
58     }
59     void init(int _s, int _t, int _vtot) {
60         s = _s;
61         t = _t;
62         vtot = _vtot;
63         etot = 0;
64         for (int i = 1; i <= vtot; i++) head[i] = -1;
65     }
66 };
```

5.7 dinic-tourist.cpp

```
1   template <typename T>
2   class flow_graph {
3    public:
4     static constexpr T eps = (T)1e-9;
5
6     struct edge {
7       int from;
8       int to;
9       T c;
10      T f;
11    };
12
13    vector<vector<int>> g;
14    vector<edge> edges;
15    int n;
16    int st;
17    int fin;
18    T flow;
19
20    flow_graph(int _n, int _st, int _fin) : n(_n), st(_st), fin(_fin) {
21      assert(0 <= st && st < n && 0 <= fin && fin < n && st != fin);
22      g.resize(n);
23      flow = 0;
24    }
25
26    void clear_flow() {
27      for (const edge &e : edges) {
28        e.f = 0;
29      }
30      flow = 0;
31    }
32
33    int add(int from, int to, T forward_cap, T backward_cap) {
34      assert(0 <= from && from < n && 0 <= to && to < n);
35      int id = (int)edges.size();
36      g[from].push_back(id);
37      edges.push_back({from, to, forward_cap, 0});
38      g[to].push_back(id + 1);
39      edges.push_back({to, from, backward_cap, 0});
40      return id;
41    }
42  };
```

```
43
44  template <typename T>
45  class dinic {
46   public:
47    flow_graph<T> &g;
48
49    vector<int> ptr;
50    vector<int> d;
51    vector<int> q;
52
53    dinic(flow_graph<T> &_g) : g(_g) {
54      ptr.resize(g.n);
55      d.resize(g.n);
56      q.resize(g.n);
57    }
58
59    bool expath() {
60      fill(d.begin(), d.end(), -1);
61      q[0] = g.fin;
62      d[g.fin] = 0;
63      int beg = 0, end = 1;
64      while (beg < end) {
65        int i = q[beg++];
66        for (int id : g.g[i]) {
67          const auto &e = g.edges[id];
68          const auto &back = g.edges[id ^ 1];
69          if (back.c - back.f > g.eps && d[e.to] == -1) {
70            d[e.to] = d[i] + 1;
71            if (e.to == g.st) {
72              return true;
73            }
74            q[end++] = e.to;
75          }
76        }
77      }
78      return false;
79    }
80
81    T dfs(int v, T w) {
82      if (v == g.fin) {
83        return w;
84      }
85      int &j = ptr[v];
```

```
86      while (j >= 0) {
87        int id = g.g[v][j];
88        const auto &e = g.edges[id];
89        if (e.c - e.f > g.eps && d[e.to] == d[v] - 1) {
90          T t = dfs(e.to, min(e.c - e.f, w));
91          if (t > g.eps) {
92            g.edges[id].f += t;
93            g.edges[id ^ 1].f -= t;
94            return t;
95          }
96        }
97        j--;
98      }
99      return 0;
100   }
101
102   T max_flow() {
103     while (expath()) {
104       for (int i = 0; i < g.n; i++) {
105         ptr[i] = (int)g.g[i].size() - 1;
106       }
107       T big_add = 0;
108       while (true) {
109         T add = dfs(g.st, numeric_limits<T>::max());
110         if (add <= g.eps) {
111           break;
112         }
113         big_add += add;
114       }
115       if (big_add <= g.eps) {
116         break;
117       }
118       g.flow += big_add;
119     }
120     return g.flow;
121   }
122
123   vector<bool> min_cut() {
124     max_flow();
125     vector<bool> ret(g.n);
126     for (int i = 0; i < g.n; i++) {
127       ret[i] = (d[i] != -1);
128     }
```

```
129     return ret;
130   }
131 };
```

5.8   eulerian-digraph.cpp

```
1   optional<vector<int>> eulerian_path(int n, const vector<PII> &E) {
2     vector<int> res;
3     if (E.empty()) return res;
4     vector<VI> adj(n + 1);
5     vector<int> in(n + 1);
6     for (int i = 0; i < ssize(E); i++) {
7       auto [u, v] = E[i];
8       adj[u].push_back(i);
9       in[v] += 1;
10    }
11
12    int s = -1, hi = 0, lo = 0;
13    for (int i = 1; i <= n; i++) {
14      if (SZ(adj[i]) == in[i]) continue;
15      if (abs(SZ(adj[i]) - in[i]) > 1) return {};
16      if (SZ(adj[i]) > in[i]) {
17        hi++, s = i;
18      } else {
19        lo++;
20      }
21    }
22    if (!(hi == 0 && lo == 0) && !(hi == 1 && lo == 1)) {
23      return {};
24    }
25    for (int i = 1; s == -1 && i <= n; i++)
26      if (!adj[i].empty()) s = i;
27
28    auto Dfs = [&](auto &Dfs, int u) -> void {
29      while (!adj[u].empty()) {
30        auto id = adj[u].back();
31        adj[u].pop_back();
32        int v = E[id].second;
33        Dfs(Dfs, v);
34        res.push_back(v);
35      }
36    };
37    Dfs(Dfs, s);
```

```
38      if (SZ(res) != SZ(E)) return {};
39      ranges::reverse(res);
40      return res;
41  }
```

5.9 eulerian-undigraph.cpp

```
1  optional<vector<int>> eulerian_path(int n, const vector<PII> &E) {
2    vector<int> res;
3    if (E.empty()) return res;
4    vector<VI> adj(n + 1);
5    for (int i = 0; i < ssize(E); i++) {
6      auto [u, v] = E[i];
7      adj[u].push_back(i);
8      adj[v].push_back(i);
9    }
10
11   int s = -1, odd = 0;
12   for (int i = 1; i <= n; i++) {
13     if (ssize(adj[i]) % 2 == 0) continue;
14     if (++odd > 2) return {};
15     s = i;
16   }
17   for (int i = 1; s == -1 && i <= n; i++)
18     if (!adj[i].empty()) s = i;
19
20   vector<int> vis(ssize(E));
21   auto Dfs = [&](auto &Dfs, int u) -> void {
22     while (!adj[u].empty()) {
23       auto id = adj[u].back();
24       adj[u].pop_back();
25       if (vis[id]) continue;
26       vis[id] = 1;
27       int v = u ^ E[id].fi ^ E[id].se;
28       Dfs(Dfs, v);
29       res.push_back(v);
30     }
31   };
32   Dfs(Dfs, s);
33   if (SZ(res) != SZ(E)) return {};
34   ranges::reverse(res);
35   return res;
36 }
```

5.10 hungarian.cpp

```
1  vector<int> g[maxn];
2  int idx;
3  int a[N][N], use[N][N], p[maxn], vis[maxn];
4
5  bool find(int x) {
6      vis[x] = 1;
7      for (auto y : g[x]) {
8          if (!p[y] || (!vis[p[y]] && find(p[y]))) {
9              p[y] = x;
10             return true;
11         }
12     }
13     return false;
14 }
15
16 int match() {
17     int res = 0;
18     fill(p + 1, p + idx + 1, 0);
19     for (int i = 1; i <= idx; i++) {
20         fill(vis + 1, vis + idx + 1, 0);
21         if (find(i)) res++;
22     }
23     return res;
24 }
```

5.11 KM.cpp

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  // L <= R, 左边完全匹配
6  // 最小权完备匹配
7
8  // 带权匹配：使得该二分图的权值和最大（或最小）的匹配。
9  // 最大匹配：使得该二分图边数最多的匹配。
10 // 完备匹配：使得点数较小的点集中每个点都被匹配的匹配。
11 // 完美匹配：所有点都被匹配的匹配。
12 // 定理1：最大匹配数 = 最小点覆盖数（Konig 定理）
13 // 定理2：最大匹配数 = 最大独立数
14 // 定理3：最小路径覆盖数 = 顶点数 - 最大匹配数
15
```

```cpp
16   // 二分图的最小点覆盖
17   // 定义：在二分图中，求最少的点集，使得每一条边至少都有端点在这个点集中。
18   // 二分图的最小点覆盖 = 二分图的最大匹配
19
20   // 二分图的最少边覆盖
21   // 定义：在二分图中，求最少的边，使得他们覆盖所有的点，并且每一个点只被一条
          边覆盖。
22   // 二分图的最少边覆盖 = 点数 - 二分图的最大匹配
23
24   // 二分图的最大独立集
25   // 定义：在二分图中，选最多的点，使得任意两个点之间没有直接边连接。
26   // 二分图的最大独立集 = 点数 - 二分图的最大匹配
27
28   template<class T>
29   pair<T, vector<int>> hungarian(const vector<vector<T>> &a) {
30       if (a.empty()) return {0, {}};
31       int n = a.size() + 1, m = a[0].size() + 1;
32       vector<T> u(n), v(m); // 顶标
33       vector<int> p(m), ans(n - 1);
34       for (int i = 1; i < n; i++) {
35           p[0] = i;
36           int j0 = 0;
37           vector<T> dist(m, numeric_limits<T>::max());
38           vector<int> pre(m, -1);
39           vector<bool> done(m + 1);
40           do { // dijkstra
41               done[j0] = true;
42               int i0 = p[j0], j1;
43               T delta = numeric_limits<T>::max();
44               for (int j = 1; j < m; j++) if (!done[j]) {
45                   auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
46                   if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
47                   if (dist[j] < delta) delta = dist[j], j1 = j;
48               }
49               for (int j = 0; j < m; j++) {
50                   if (done[j]) u[p[j]] += delta, v[j] -= delta;
51                   else dist[j] -= delta;
52               }
53               j0 = j1;
54           } while (p[j0]);
55           while (j0) { // update alternating path
56               int j1 = pre[j0];
57               p[j0] = p[j1], j0 = j1;
58           }
59       }
60       for (int j = 1; j < m; j++) {
61           if (p[j]) ans[p[j] - 1] = j - 1;
62       }
63       return {-v[0], ans}; // min cost
64   }
65
66   int L, R, m;
67   int main() {
68       scanf("%d%d%d", &L, &R, &m);
69       R = max(L, R);
70       auto a = vector<vector<ll>>(L, vector<ll>(R, 0));
71       for (int i = 0; i < m; i++) {
72           int u, v, w;
73           scanf("%d%d%d", &u, &v, &w);
74           --u; --v;
75           a[u][v] = -w;
76       }
77       auto [val, ans] = hungarian(a);
78       printf("%lld\n", -val);
79       for (int i = 0; i < L; i++) {
80           if (a[i][ans[i]] >= 0) ans[i] = -1;
81           printf("%d%c", ans[i] + 1, " \n"[i == L - 1]);
82       }
83   }
```

5.12 kosaraju.cpp

```cpp
1   vector<int> e[maxn], erev[maxn];
2   vector<int> c, out;
3   vector<vector<int>> scc;
4   int vis[maxn];
5   void dfs(int u) {
6       vis[u] = 1;
7       for (auto v : e[u]) if (!vis[v]) dfs(v);
8       out.pb(u);
9   }
10  void dfs_rev(int u) {
11      vis[u] = 1;
12      for (auto v : erev[u]) if (!vis[v]) dfs_rev(v);
13      c.pb(u);
14  }
```

```
15    void solve() {
16        cin >> n >> m;
17        rep(i, 1, m) {
18            int u, v;
19            cin >> u >> v;
20            e[u].pb(v);
21            erev[v].pb(u);
22        }
23        rep(i, 1, n) if (!vis[i]) dfs(i);
24        fill(vis + 1, vis + n + 1, 0);
25        reverse(all(out));
26        for (auto v : out) if (!vis[v]) {
27            c.clear();
28            dfs_rev(v);
29            scc.pb(c);
30        }
31    }
```

5.13   kruskal 重构树.cpp

```
1    /**
2     * 构建后是一颗二叉树，如果按最小生成树建立的话是大根堆。
3     * 性质：原图中两个点间所有路径上的边最大权值的最小值=最小生成树上两点简单路
              径的边最大权值
4     * =kruskal 重构树上两点 LCA 的权值。
5     * 重构树中代表原树中的点的节点全是叶子节点，其余节点都代表了一条边的边权。
6     * 利用这个性质可以找到点 P 的简单路径上边权最大值小于 lim 深度最小的节点。
7     * 要求最小权值最大值，可以建最大生成树的重构树从而达到一样的效果。
8     */
9
10   vector<tuple<ll, ll, ll>> E;
11   rep(i, 1, m) {
12        int u, v, w;
13        cin >> u >> v >> w;
14        E.emplace_back(w, u, v);
15   }
16   ranges::sort(E);
17   for (auto [w, u, v] : E) {
18        u = find(u), v = find(v);
19        if (u == v) continue;
20        int p = ++idx;
21        lim[p] = w;
22        fa[u] = p, fa[v] = p;
```

```
23            e[p].push_back(u);
24            e[u].push_back(p);
25            e[p].push_back(v);
26            e[v].push_back(p);
27    }
```

5.14   MCMF.cpp

```
1    template<typename T>
2    struct MinCostGraph {
3        static const int V = 20100;
4        static const int E = 201000;
5        int s, t, vtot;
6        int head[V], etot;
7        T dis[V], flow, cost;
8        int pre[V];
9        bool vis[V];
10
11        struct edge {
12            int v, nxt;
13            T f, c;
14        } e[E * 2];
15        void addedge(int u,int v, T f, T c, T f2 = 0){
16            e[etot]= {v, head[u], f, c}; head[u] = etot++;
17            e[etot]= {u, head[v], f2, -c}; head[v] = etot++;
18        }
19
20        bool spfa() {
21            T inf = numeric_limits<T>::max() / 2;
22            for (int i = 1; i <= vtot; i++) {
23                dis[i] = inf;
24                vis[i] = false;
25                pre[i] = -1;
26            }
27            dis[s] = 0;
28            vis[s] = true;
29            queue<int> q;
30            q.push(s);
31            while (!q.empty()) {
32                int u = q.front();
33                for (int i = head[u]; ~i; i = e[i].nxt) {
34                    int v = e[i].v;
35                    if (e[i].f && dis[v] > dis[u] + e[i].c) {
```

```
36                          dis[v] = dis[u] + e[i].c;
37                          pre[v] = i;
38                          if (!vis[v]) {
39                              vis[v] = 1;
40                              q.push(v);
41                          }
42                      }
43                  }
44              q.pop();
45              vis[u] = false;
46          }
47          return dis[t] != inf;
48      }
49
50      void augment() {
51          int u = t;
52          T f = numeric_limits<T>::max();
53          while (~pre[u]) {
54              f = min(f, e[pre[u]].f);
55              u = e[pre[u] ^ 1].v;
56          }
57          flow += f;
58          cost += f * dis[t];
59          u = t;
60          while (~pre[u]) {
61              e[pre[u]].f -= f;
62              e[pre[u] ^ 1].f += f;
63              u = e[pre[u] ^ 1].v;
64          }
65      }
66
67      pair<T, T> solve() {
68          flow = 0;
69          cost = 0;
70          while (spfa()) augment();
71          return {flow, cost};
72      }
73      void init(int s_, int t_, int vtot_) {
74          s = s_;
75          t = t_;
76          vtot = vtot_;
77          etot = 0;
78          for (int i = 1; i <= vtot; i++) head[i] = -1;
```

```
79      }
80  };
```

5.15 MCMFfast.cpp

```
1   template <typename flow_t = int, typename cost_t = long long>
2   struct MCMF_SSPA {
3       int N;
4       vector<vector<int>> adj;
5       struct edge_t {
6           int dest;
7           flow_t cap;
8           cost_t cost;
9       };
10      vector<edge_t> edges;
11
12      vector<char> seen;
13      vector<cost_t> pi;
14      vector<int> prv;
15
16      explicit MCMF_SSPA(int N_) : N(N_), adj(N), pi(N, 0), prv(N) {}
17
18      void addEdge(int from, int to, flow_t cap, cost_t cost) {
19          assert(cap >= 0);
20          int e = int(edges.size());
21          edges.emplace_back(edge_t{to, cap, cost});
22          edges.emplace_back(edge_t{from, 0, -cost});
23          adj[from].push_back(e);
24          adj[to].push_back(e+1);
25      }
26
27      const cost_t INF_COST = numeric_limits<cost_t>::max() / 4;
28      const flow_t INF_FLOW = numeric_limits<flow_t>::max() / 4;
29      vector<cost_t> dist;
30      __gnu_pbds::priority_queue<pair<cost_t, int>> q;
31      vector<typename decltype(q)::point_iterator> its;
32      void path(int s) {
33          dist.assign(N, INF_COST);
34          dist[s] = 0;
35
36          its.assign(N, q.end());
37          its[s] = q.push({0, s});
38
```

```
39         while (!q.empty()) {
40             int i = q.top().second; q.pop();
41             cost_t d = dist[i];
42             for (int e : adj[i]) {
43                 if (edges[e].cap) {
44                     int j = edges[e].dest;
45                     cost_t nd = d + edges[e].cost;
46                     if (nd < dist[j]) {
47                         dist[j] = nd;
48                         prv[j] = e;
49                         if (its[j] == q.end()) {
50                             its[j] = q.push({-(dist[j] - pi[j]), j});
51                         } else {
52                             q.modify(its[j], {-(dist[j] - pi[j]), j});
53                         }
54                     }
55                 }
56             }
57         }
58
59         swap(pi, dist);
60     }
61
62     vector<pair<flow_t, cost_t>> maxflow(int s, int t) {
63         assert(s != t);
64         flow_t totFlow = 0; cost_t totCost = 0;
65         vector<pair<flow_t, cost_t>> res;
66         while (path(s), pi[t] < INF_COST) {
67             flow_t curFlow = numeric_limits<flow_t>::max();
68             for (int cur = t; cur != s; ) {
69                 int e = prv[cur];
70                 int nxt = edges[e^1].dest;
71                 curFlow = min(curFlow, edges[e].cap);
72                 cur = nxt;
73             }
74             totFlow += curFlow;
75             totCost += pi[t] * curFlow;
76             for (int cur = t; cur != s; ) {
77                 int e = prv[cur];
78                 int nxt = edges[e^1].dest;
79                 edges[e].cap -= curFlow;
80                 edges[e^1].cap += curFlow;
81                 cur = nxt;
```

```
82             }
83
84             res.emplace_back(totFlow, totCost);
85         }
86         return res;
87     }
88 };
```

5.16   MCMFfull.cpp

```
1  template <typename T, typename C>
2  class MCMF {
3   public:
4    static constexpr T eps = (T) 1e-9;
5
6    struct edge {
7      int from;
8      int to;
9      T c;
10     T f;
11     C cost;
12   };
13
14   int n;
15   vector<vector<int>> g;
16   vector<edge> edges;
17   vector<C> d;
18   vector<C> pot;
19   __gnu_pbds::priority_queue<pair<C, int>> q;
20   vector<typename decltype(q)::point_iterator> its;
21   vector<int> pe;
22   const C INF_C = numeric_limits<C>::max() / 2;
23
24   explicit MCMF(int n_) : n(n_), g(n), d(n), pot(n, 0), its(n), pe(n) {}
25
26   int add(int from, int to, T forward_cap, T backward_cap, C edge_cost) {
27     assert(0 <= from && from < n && 0 <= to && to < n);
28     assert(forward_cap >= 0 && backward_cap >= 0);
29     int id = static_cast<int>(edges.size());
30     g[from].push_back(id);
31     edges.push_back({from, to, forward_cap, 0, edge_cost});
32     g[to].push_back(id + 1);
33     edges.push_back({to, from, backward_cap, 0, -edge_cost});
```

```
34        return id;
35      }
36
37      void expath(int st) {
38        fill(d.begin(), d.end(), INF_C);
39        q.clear();
40        fill(its.begin(), its.end(), q.end());
41        its[st] = q.push({pot[st], st});
42        d[st] = 0;
43        while (!q.empty()) {
44          int i = q.top().second;
45          q.pop();
46          its[i] = q.end();
47          for (int id : g[i]) {
48            const edge &e = edges[id];
49            int j = e.to;
50            if (e.c - e.f > eps && d[i] + e.cost < d[j]) {
51              d[j] = d[i] + e.cost;
52              pe[j] = id;
53              if (its[j] == q.end()) {
54                its[j] = q.push({pot[j] - d[j], j});
55              } else {
56                q.modify(its[j], {pot[j] - d[j], j});
57              }
58            }
59          }
60        }
61        swap(d, pot);
62      }
63
64      pair<T, C> calc(int st, int fin) {  // max_flow_min_cost
65        T flow = 0;
66        C cost = 0;
67        bool ok = true;
68        for (auto& e : edges) {
69          if (e.c - e.f > eps && e.cost + pot[e.from] - pot[e.to] < 0) {
70            ok = false;
71            break;
72          }
73        }
74        if (ok) {
75          expath(st);
76        } else {
77          vector<int> deg(n, 0);
78          for (int i = 0; i < n; i++) {
79            for (int eid : g[i]) {
80              auto& e = edges[eid];
81              if (e.c - e.f > eps) {
82                deg[e.to] += 1;
83              }
84            }
85          }
86          vector<int> que;
87          for (int i = 0; i < n; i++) {
88            if (deg[i] == 0) {
89              que.push_back(i);
90            }
91          }
92          for (int b = 0; b < (int) que.size(); b++) {
93            for (int eid : g[que[b]]) {
94              auto& e = edges[eid];
95              if (e.c - e.f > eps) {
96                deg[e.to] -= 1;
97                if (deg[e.to] == 0) {
98                  que.push_back(e.to);
99                }
100             }
101           }
102         }
103         fill(pot.begin(), pot.end(), INF_C);
104         pot[st] = 0;
105         if (static_cast<int>(que.size()) == n) {
106           for (int v : que) {
107             if (pot[v] < INF_C) {
108               for (int eid : g[v]) {
109                 auto& e = edges[eid];
110                 if (e.c - e.f > eps) {
111                   if (pot[v] + e.cost < pot[e.to]) {
112                     pot[e.to] = pot[v] + e.cost;
113                     pe[e.to] = eid;
114                   }
115                 }
116               }
117             }
118           }
119         } else {
```

```
120          que.assign(1, st);
121          vector<bool> in_queue(n, false);
122          in_queue[st] = true;
123          for (int b = 0; b < (int) que.size(); b++) {
124            int i = que[b];
125            in_queue[i] = false;
126            for (int id : g[i]) {
127              const edge &e = edges[id];
128              if (e.c - e.f > eps && pot[i] + e.cost < pot[e.to]) {
129                pot[e.to] = pot[i] + e.cost;
130                pe[e.to] = id;
131                if (!in_queue[e.to]) {
132                  que.push_back(e.to);
133                  in_queue[e.to] = true;
134                }
135              }
136            }
137          }
138        }
139      }
140      // debug(pot[fin]);
141      while (pot[fin] < INF_C) {  // < 0
142        T push = numeric_limits<T>::max();
143        int v = fin;
144        while (v != st) {
145          const edge &e = edges[pe[v]];
146          push = min(push, e.c - e.f);
147          v = e.from;
148        }
149        v = fin;
150        while (v != st) {
151          edge &e = edges[pe[v]];
152          e.f += push;
153          edge &back = edges[pe[v] ^ 1];
154          back.f -= push;
155          v = e.from;
156        }
157        flow += push;
158        cost += push * pot[fin];
159        expath(st);
160      }
161      return {flow, cost};
162    }
```

```
163  };
```

5.17   prim.cpp

```
1  vector<PII> e[N];
2
3  template <typename T>
4  void add(int u, int v, T w) {
5      e[u].eb(v, w);
6  }
7
8  template <typename T>
9  T prim(vector<pair<int, T>> *g, int start) {
10     const T inf = numeric_limits<T>::max() / 4;
11     T res = 0;
12     vector<T> dist(n, inf);
13     dist[start] = 0;
14     priority_queue<pair<T, int>, vector<pair<T, int>>, greater<pair<T, int
          >>> s;
15     s.emplace(dist[start], start);
16     vector<int> was(n, 0);
17     while (!s.empty()) {
18         T expected = s.top().first;
19         int i = s.top().second;
20         s.pop();
21         if (dist[i] != expected || was[i]) {
22             continue;
23         }
24         was[i] = 1;
25         res += expected;
26         for (auto [to, cost] : g[i]) {
27             if (cost < dist[to]) {
28                 dist[to] = cost;
29                 s.emplace(dist[to], to);
30             }
31         }
32     }
33     return res;
34  }
```

5.18   PushRelabel.cpp

```
1  /**
```

```
2   * Author: Simon Lindholm
3   * Date: 2015-02-24
4   * License: CC0
5   * Source: Wikipedia, tinyKACTL
6   * Description: Push-relabel using the highest label selection rule and the
        gap heuristic. Quite fast in practice.
7   *  To obtain the actual flow, look at positive values only.
8   * Time: $O(V^2\sqrt E)$
9   * Status: Tested on Kattis and SPOJ, and stress-tested
10  */
11  #pragma once
12
13  struct PushRelabel {
14    typedef vector<int> vi;
15    struct Edge {
16      int dest, back;
17      ll f, c;
18    };
19    vector<vector<Edge>> g;
20    vector<ll> ec;
21    vector<Edge*> cur;
22    vector<vi> hs; vi H;
23    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
24
25    void addEdge(int s, int t, ll cap, ll rcap=0) {
26      if (s == t) return;
27      g[s].push_back({t, SZ(g[t]), 0, cap});
28      g[t].push_back({s, SZ(g[s])-1, 0, rcap});
29    }
30
31    void addFlow(Edge& e, ll f) {
32      Edge &back = g[e.dest][e.back];
33      if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
34      e.f += f; e.c -= f; ec[e.dest] += f;
35      back.f -= f; back.c += f; ec[back.dest] -= f;
36    }
37    ll calc(int s, int t) {
38      int v = SZ(g); H[s] = v; ec[t] = 1;
39      vi co(2*v); co[0] = v-1;
40      rep(i,0,v-1) cur[i] = g[i].data();
41      for (Edge& e : g[s]) addFlow(e, e.c);
42
43      for (int hi = 0;;) {
```

```
44        while (hs[hi].empty()) if (!hi--) return -ec[s];
45        int u = hs[hi].back(); hs[hi].pop_back();
46        while (ec[u] > 0)  // discharge u
47          if (cur[u] == g[u].data() + SZ(g[u])) {
48            H[u] = 1e9;
49            for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
50              H[u] = H[e.dest]+1, cur[u] = &e;
51            if (++co[H[u]], !--co[hi] && hi < v)
52              rep(i,0,v-1) if (hi < H[i] && H[i] < v)
53                --co[H[i]], H[i] = v + 1;
54            hi = H[u];
55          } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
56            addFlow(*cur[u], min(ec[u], cur[u]->c));
57          else ++cur[u];
58      }
59    }
60    bool leftOfMinCut(int a) { return H[a] >= SZ(g); }
61  };
```

5.19   tarjan 割点.cpp

```
1   vector<int> g[maxn], ans;
2   stack<int> stk;
3   int dfn[maxn], cut[maxn], low[maxn], idx;
4
5   void dfs(int x, int f) {
6       low[x] = dfn[x] = ++idx;
7       stk.push(x);
8       int ch = 0;
9       for (auto y : g[x]) {
10          if (!dfn[y]) {
11              ch++;
12              dfs(y, x);
13              low[x] = min(low[x], low[y]);
14              if (low[y] >= dfn[x]) cut[x] = 1;
15          } else {
16              if (y != f) low[x] = min(low[x], dfn[y]);
17          }
18      }
19      if (x == 1 && ch <= 1) cut[x] = 0;
20      if (cut[x]) ans.pb(x);
21  }
```

5.20 tarjan 割边.cpp

```cpp
vector<PII> g[maxn];
stack<int> stk;
int dfn[maxn], ins[maxn], low[maxn];
int idx, tot;
VI ans;
void dfs(int x, int f) {
    low[x] = dfn[x] = ++idx;
    stk.push(x);
    ins[x] = 1;
    for (auto [y, id] : g[x]) {
        if (!dfn[y]) {
            dfs(y, id);
            low[x] = min(low[x], low[y]);
        } else {
            if (ins[y] && id != f) low[x] = min(low[x], dfn[y]);
        }
    }
    if (low[x] >= dfn[x]) {
        ++tot;
        while (true) {
            int cur = stk.top();
            stk.pop();
            ins[cur] = 0;
            if (cur == x) break;
        }
        if (f != 0) ans.pb(f);
    }
}
```

5.21 tarjan 强连通分量.cpp

```cpp
vector<int> g[maxn];
stack<int> stk;
int dfn[maxn], ins[maxn], low[maxn], belong[maxn];
int idx, tot;

void dfs(int x) {
    low[x] = dfn[x] = ++idx;
    ins[x] = 1;
    stk.push(x);
    for (auto y : g[x]) {
        if (!dfn[y]) {
```

```cpp
            dfs(y);
            low[x] = min(low[x], low[y]);
        } else {
            if (ins[y]) low[x] = min(low[x], dfn[y]);
        }
    }
    if (low[x] >= dfn[x]) {
        ++tot;
        while (true) {
            int cur = stk.top(); stk.pop();
            ins[cur] = 0;
            belong[cur] = tot;
            if (cur == x) break;
        }
    }
}
```

5.22 tarjan 点双.cpp

```cpp
vector<int> g[maxn];
stack<int> stk;
int dfn[maxn], low[maxn], idx, tot, cut[maxn];
vector<int> bcc[maxn];

void dfs(int x, int f) {
    low[x] = dfn[x] = ++idx;
    stk.push(x);
    int ch = 0;
    for (auto y : g[x]) {
        if (!dfn[y]) {
            ch++;
            dfs(y, x);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x]) {
                cut[x] = 1;
                ++tot;
                bcc[tot].pb(x);
                while (true) {
                    int cur = stk.top();
                    stk.pop();
                    bcc[tot].pb(cur);
                    if (cur == y) break;
                }
```

```
25                }
26            } else {
27                if (y != f) low[x] = min(low[x], dfn[y]);
28            }
29        }
30        if (x == 1 && ch <= 1) cut[x] = 0;
31    }
```

5.23   tarjan 边双.cpp

```
1  vector<PII> g[maxn];
2  stack<int> stk;
3  int dfn[maxn], low[maxn], idx, tot, belong[maxn];
4  vector<int> bcc[maxn];
5
6  void dfs(int x, int f) {
7      low[x] = dfn[x] = ++idx;
8      stk.push(x);
9      for (auto [y, id] : g[x]) {
10         if (!dfn[y]) {
11             dfs(y, id);
12             low[x] = min(low[x], low[y]);
13         } else {
14             if (id != f) low[x] = min(low[x], dfn[y]);
15         }
16     }
17     if (low[x] >= dfn[x]) {
18         ++tot;
19         while (true) {
20             int cur = stk.top();
21             stk.pop();
22             belong[cur] = tot;
23             bcc[tot].pb(cur);
24             if (cur == x) break;
25         }
26     }
27 }
```

5.24   twosat.cpp

```
1  class twosat {
2  public:
3      digraph<int> g;
```

```
4      int n;
5
6      twosat(int _n) : g(digraph<int>(2 * _n)), n(_n) {
7      }
8
9      // (v[x] == value_x)
10     inline void add(int x, int value_x) {
11         assert(0 <= x && x < n);
12         assert(0 <= value_x && value_x <= 1);
13         g.add(2 * x + (value_x ^ 1), 2 * x + value_x);
14     }
15
16     // (v[x] == value_x || v[y] == value_y)
17     inline void add(int x, int value_x, int y, int value_y) {
18         assert(0 <= x && x < n && 0 <= y && y < n);
19         assert(0 <= value_x && value_x <= 1 && 0 <= value_y && value_y <= 1)
                   ;
20         g.add(2 * x + (value_x ^ 1), 2 * y + value_y);
21         g.add(2 * y + (value_y ^ 1), 2 * x + value_x);
22     }
23
24     inline vector<int> solve() {
25         int cnt;
26         vector<int> c = find_scc(g, cnt);
27         vector<int> res(n);
28         for (int i = 0; i < n; i++) {
29             if (c[2 * i] == c[2 * i + 1]) {
30                 return vector<int>();
31             }
32             res[i] = (c[2 * i] < c[2 * i + 1]);
33         }
34         return res;
35     }
36 };
```

5.25   差分约束系统.cpp

```
1  /**
2     Description:
3     求解方程组 x_u - x_v <= w_i, 求出的x_i为满足条件的最大值
4      转化为x_u <= x_v + w_i
5     问题等价于求最短路 (bellmanford或Floyd)
6     即加一条有向边add(u, v, w), dist[v] = min(dist[v], dist[u] + w)
```

```
7        求 最 小 值 （ 满 足 条 件 情 况 下 尽 量 小 ） 等 价 于 求 (-x_i) 最 大 （ 或 者 转 化 为 求 最 长 路 ）
8          求 非 负 解 只 需 要 添 加 超 级 节 点 S， S 向 各 个 点 连 边 （ S + 0 <= x_i ）， 再 设 dist[S]
                = 0
9    */
10   void solve() {
11       cin >> n >> m;
12       vector<int> dist(n, 0);
13       vector<vector<PII>> g(n);
14       rep(i, 0, m - 1) {
15           int u, v, w;
16           cin >> u >> v >> w;
17           u--, v--;
18           g[u].eb(v, -w);
19       }
20       bool ok = 1;
21       rep(i, 1, n) {
22           bool upd = 0;
23           rep(u, 0, n - 1) {
24               for (auto [v, w] : g[u]) {
25                   if (dist[v] < dist[u] + w) {
26                       dist[v] = dist[u] + w;
27                       upd = 1;
28                   }
29               }
30           }
31           if (!upd) break;
32           // 仍 然 有 约 束 未 满 足
33           if (i == n && upd) ok = 0;
34       }
35       if (!ok) {
36           return cout << -1 << '\n', void();
37       }
38       rep(i, 0, n - 1) {
39           cout << dist[i] << "␣\n"[i == n - 1];
40       }
41   }
```

6 Math

6.1 binom.cpp

```
1    vector<Mint> fact(1, 1);
```

```
2    vector<Mint> inv_fact(1, 1);
3
4    Mint C(int n, int k) {
5        if (k < 0 || k > n) {
6            return 0;
7        }
8        while ((int)fact.size() < n + 1) {
9            fact.push_back(fact.back() * (int)fact.size());
10           inv_fact.push_back(1 / fact.back());
11       }
12       return fact[n] * inv_fact[k] * inv_fact[n - k];
13   }
14
15   const int mod = 1000000007;
16   const int T = 1000000;
17   ll fact[] = {};
18   ll powmod(ll a, ll b) {
19       ll ret = 1;
20       for (; b; b >>= 1) {
21           if (b & 1) ret = ret * a % mod;
22           a = a * a %mod;
23       }
24       return ret;
25   }
26   ll fac(int n) {
27       ll v = fact[n / T];
28       for (int i = n / T * T + 1; i <= n; i++)
29           v = v * i % mod;
30       return v;
31   }
32   ll binom(int n, int m) {
33       if (m < 0 || m > n) return 0;
34       return fac(n) * powmod(fac(m) * fac(n - m) % mod, mod - 2) % mod;
35   }
```

6.2 bsgs.cpp

```
1    int bsgs(int a, int b, int m) { // a^x=b(mod m)
2        int res = m + 1;
3        int t = sqrt(m) + 2;
4        ll d = powmod(a, t, m);
5        ll cnt = 1;
6        //map<int,int> p;
```

```
 7        hs.init();
 8        for (int i = 1; i <= t; i++) {
 9            cnt = cnt * d % m;
10            //if (!p.count(cnt)) p[cnt] = i;
11            if (hs.query(cnt) == -1) hs.insert(cnt, i);
12        }
13        cnt = b;
14        for (int i = 1; i <= t; i++) {
15            cnt = cnt * a % m;
16            //if (p.count(cnt)) res = min(res, p[cnt] * t - i);
17            int tmp = hs.query(cnt);
18            if (tmp != -1) res = min(res, tmp * t - i);
19        }
20        if (res >= m) res = -1;
21        return res;
22    }
```

6.3   cantor.cpp

```
 1  ll fac[maxn], A[maxn], w[maxn];
 2  void init(int n) {
 3      fac[0] = 1;
 4      rep(i, 1, n) fac[i] = fac[i - 1] * i % mod;
 5  }
 6  ll cantor(int w[], int n) {
 7      ll ans = 1;
 8      for (int i = 1; i <= n; i++) { // can optimize by BIT
 9          for (int j = i + 1; j <= n; j++) {
10              if (w[i] > w[j]) A[i]++;
11          }
12      }
13      for (int i = 1; i < n; i++) {
14          ans += A[i] * fac[n - i];
15      }
16      return ans;
17  }
18
19  void decanter(ll x, int n) { // x->rank n->length
20      x--;
21      vector<int> rest(n, 0);
22      iota(rest.begin(), rest.end(), 1); // rest->1,2,3,4...
23      for (int i = 1; i <= n; i++) {
24          A[i] = x / fac[n - i];
```

```
25          x %= fac[n - i];
26      }
27      for (int i = 1; i <= n; i++) {
28          w[i] = rest[A[i]];
29          rest.erase(lower_bound(rest.begin(), rest.end(), w[i]));
30      }
31  }
```

6.4   EXCRT modequ exgcd.cpp

```
 1  ll exgcd(ll a, ll b, ll &x, ll &y) {
 2      if (b == 0) {
 3          x = 1, y = 0;
 4          return a;
 5      }
 6      ll d = exgcd(b, a % b, y, x);
 7      y -= (a / b) * x;
 8      return d;
 9  }
10
11  // 求 a * x = b (mod m) 的解
12  ll modequ(ll a, ll b, ll m) {
13      ll x, y;
14      ll d = exgcd(a, m, x, y);
15      if (b % d != 0) return -1;
16      m /= d; a /= d; b /= d;
17      x = x * b % m;
18      if (x < 0) x += m;
19      return x;
20  }
21
22  void merge(ll &a, ll &b, ll c, ll d) {
23      if (a == -1 || b == -1) return;
24      ll x, y;
25      ll g = exgcd(b, d, x, y);
26      if ((c - a) % g != 0) {
27          a = -1, b = -1;
28          return;
29      }
30      d /= g;
31      ll t = ((c - a) / g) % d * x % d;
32      if (t < 0) t += d;
33      a = b * t + a;
```

```
34        b = b * d;
35    }
```

6.5 factor.cpp

```
1   namespace Factor {
2       const int N=1010000;
3       ll C,fac[10010],n,mut,a[1001000];
4       int T,cnt,i,l,prime[N],p[N],psize,_cnt;
5       ll _e[100],_pr[100];
6       vector<ll> d;
7       inline ll mul(ll a,ll b,ll p) {
8           if (p<=1000000000) return a*b%p;
9           else if (p<=100000000000ll) return (((a*(b>>20)%p)<<20)+(a*(b
                &((1<<20)-1))))%p;
10          else {
11              ll d=(ll)floor(a*(long double)b/p+0.5);
12              ll ret=(a*b-d*p)%p;
13              if (ret<0) ret+=p;
14              return ret;
15          }
16      }
17      void prime_table(){
18          int i,j,tot,t1;
19          for (i=1;i<=psize;i++) p[i]=i;
20          for (i=2,tot=0;i<=psize;i++){
21              if (p[i]==i) prime[++tot]=i;
22              for (j=1;j<=tot && (t1=prime[j]*i)<=psize;j++){
23                  p[t1]=prime[j];
24                  if (i%prime[j]==0) break;
25              }
26          }
27      }
28      void init(int ps) {
29          psize=ps;
30          prime_table();
31      }
32      ll powl(ll a,ll n,ll p) {
33          ll ans=1;
34          for (;n;n>>=1) {
35              if (n&1) ans=mul(ans,a,p);
36              a=mul(a,a,p);
37          }
```

```
38          return ans;
39      }
40      bool witness(ll a,ll n) {
41          int t=0;
42          ll u=n-1;
43          for (;~u&1;u>>=1) t++;
44          ll x=powl(a,u,n),_x=0;
45          for (;t;t--) {
46              _x=mul(x,x,n);
47              if (_x==1 && x!=1 && x!=n-1) return 1;
48              x=_x;
49          }
50          return _x!=1;
51      }
52      bool miller(ll n) {
53          if (n<2) return 0;
54          if (n<=psize) return p[n]==n;
55          if (~n&1) return 0;
56          for (int j=0;j<=7;j++) if (witness(rng()%(n-1)+1,n)) return 0;
57          return 1;
58      }
59      ll gcd(ll a,ll b) {
60          ll ret=1;
61          while (a!=0) {
62              if ((~a&1) && (~b&1)) ret<<=1,a>>=1,b>>=1;
63              else if (~a&1) a>>=1; else if (~b&1) b>>=1;
64              else {
65                  if (a<b) swap(a,b);
66                  a-=b;
67              }
68          }
69          return ret*b;
70      }
71      ll rho(ll n) {
72          while (1) {
73              ll X=rng()%n,Y,Z,T=1,*lY=a,*lX=lY;
74              int tmp=20;
75              C=rng()%10+3;
76              X=mul(X,X,n)+C;*(lY++)=X;lX++;
77              Y=mul(X,X,n)+C;*(lY++)=Y;
78              for (;X!=Y;) {
79                  ll t=X-Y+n;
80                  Z=mul(T,t,n);
```

```
 81                 if(Z==0) return gcd(T,n);
 82             tmp--;
 83             if (tmp==0) {
 84                 tmp=20;
 85                 Z=gcd(Z,n);
 86                 if (Z!=1 && Z!=n) return Z;
 87             }
 88             T=Z;
 89             Y=*(lY++)=mul(Y,Y,n)+C;
 90             Y=*(lY++)=mul(Y,Y,n)+C;
 91             X=*(lX++);
 92         }
 93     }
 94 }
 95 void _factor(ll n) {
 96     for (int i=0;i<cnt;i++) {
 97         if (n%fac[i]==0) n/=fac[i],fac[cnt++]=fac[i];}
 98     if (n<=psize) {
 99         for (;n!=1;n/=p[n]) fac[cnt++]=p[n];
100         return;
101     }
102     if (miller(n)) fac[cnt++]=n;
103     else {
104         ll x=rho(n);
105         _factor(x);_factor(n/x);
106     }
107 }
108 void dfs(ll x,int dep) {
109     if (dep==_cnt) d.pb(x);
110     else {
111         dfs(x,dep+1);
112         for (int i=1;i<=_e[dep];i++) dfs(x*=_pr[dep],dep+1);
113     }
114 }
115 void norm() {
116     sort(fac,fac+cnt);
117     _cnt=0;
118     rep(i,0,cnt-1) if (i==0||fac[i]!=fac[i-1]) _pr[_cnt]=fac[i],_e[_cnt
            ++]=1;
119     else _e[_cnt-1]++;
120 }
121 vector<ll> getd() {
122     d.clear();
```

```
123         dfs(1,0);
124         return d;
125     }
126     vector<ll> factor(ll n) {
127         cnt=0;
128         _factor(n);
129         norm();
130         return getd();
131     }
132     vector<PLL> factorG(ll n) {
133         cnt=0;
134         _factor(n);
135         norm();
136         vector<PLL> d;
137         rep(i,0,_cnt-1) d.pb(mp(_pr[i],_e[i]));
138         return d;
139     }
140     bool is_primitive(ll a,ll p) {
141         assert(miller(p));
142         vector<PLL> D=factorG(p-1);
143         rep(i,0,SZ(D)-1) if (powl(a,(p-1)/D[i].fi,p)==1) return 0;
144         return 1;
145     }
146     ll phi(ll n) {
147         auto d=factorG(n);
148         for (auto p:d) n=n/p.fi*(p.fi-1);
149         return n;
150     }
151 }
```

6.6   fft.cpp

```
 1 namespace fft {
 2   typedef double dbl;
 3
 4   struct num {
 5     dbl x, y;
 6     num() { x = y = 0; }
 7     num(dbl x, dbl y) : x(x), y(y) { }
 8   };
 9
10   inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
11   inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
```

```
12      inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a.x
            * b.y + a.y * b.x); }
13      inline num conj(num a) { return num(a.x, -a.y); }
14
15      int base = 1;
16      vector<num> roots = {{0, 0}, {1, 0}};
17      vector<int> rev = {0, 1};
18
19      const dbl PI = acosl(-1.0);
20
21      void ensure_base(int nbase) {
22        if (nbase <= base) {
23          return;
24        }
25        rev.resize(1 << nbase);
26        for (int i = 0; i < (1 << nbase); i++) {
27          rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
28        }
29        roots.resize(1 << nbase);
30        while (base < nbase) {
31          dbl angle = 2 * PI / (1 << (base + 1));
32 //         num z(cos(angle), sin(angle));
33          for (int i = 1 << (base - 1); i < (1 << base); i++) {
34            roots[i << 1] = roots[i];
35 //           roots[(i << 1) + 1] = roots[i] * z;
36            dbl angle_i = angle * (2 * i + 1 - (1 << base));
37            roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
38          }
39          base++;
40        }
41      }
42
43      void fft(vector<num> &a, int n = -1) {
44        if (n == -1) {
45          n = a.size();
46        }
47        assert((n & (n - 1)) == 0);
48        int zeros = __builtin_ctz(n);
49        ensure_base(zeros);
50        int shift = base - zeros;
51        for (int i = 0; i < n; i++) {
52          if (i < (rev[i] >> shift)) {
53            swap(a[i], a[rev[i] >> shift]);

54          }
55        }
56 /*      for (int k = 1; k < n; k <<= 1) {
57        for (int i = 0; i < n; i += 2 * k) {
58          for (int j = 0; j < k; j++) {
59            num z = a[i + j + k] * roots[j + k];
60            a[i + j + k] = a[i + j] - z;
61            a[i + j] = a[i + j] + z;
62          }
63        }
64      }*/
65        for (int len = 1; len < n; len <<= 1) {
66          for (int i = 0; i < n; i += 2 * len) {
67            for (int j = i, k = i + len; j < i + len; j++, k++) {
68              num z = a[k] * roots[k - i];
69              a[k] = a[j] - z;
70              a[j] = a[j] + z;
71            }
72          }
73        }
74      }
75
76      vector<num> fa, fb;
77
78      vector<long long> multiply(vector<int> &a, vector<int> &b) {
79        int need = a.size() + b.size() - 1;
80        int nbase = 0;
81        while ((1 << nbase) < need) nbase++;
82        ensure_base(nbase);
83        int sz = 1 << nbase;
84        if (sz > (int) fa.size()) {
85          fa.resize(sz);
86        }
87        for (int i = 0; i < sz; i++) {
88          int x = (i < (int) a.size() ? a[i] : 0);
89          int y = (i < (int) b.size() ? b[i] : 0);
90          fa[i] = num(x, y);
91        }
92        fft(fa, sz);
93        num r(0, -0.25 / sz);
94        for (int i = 0; i <= (sz >> 1); i++) {
95          int j = (sz - i) & (sz - 1);
96          num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
```

```
97        if (i != j) {
98          fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
99        }
100       fa[i] = z;
101     }
102     fft(fa, sz);
103     vector<long long> res(need);
104     for (int i = 0; i < need; i++) {
105       res[i] = fa[i].x + 0.5;
106     }
107     return res;
108   }
109
110   vector<int> multiply_mod(vector<int> &a, vector<int> &b, int m, int eq =
          0) {
111     int need = a.size() + b.size() - 1;
112     int nbase = 0;
113     while ((1 << nbase) < need) nbase++;
114     ensure_base(nbase);
115     int sz = 1 << nbase;
116     if (sz > (int) fa.size()) {
117       fa.resize(sz);
118     }
119     for (int i = 0; i < (int) a.size(); i++) {
120       int x = (a[i] % m + m) % m;
121       fa[i] = num(x & ((1 << 15) - 1), x >> 15);
122     }
123     fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
124     fft(fa, sz);
125     if (eq) {
126       copy(fa.begin(), fa.begin() + sz, fb.begin());
127     } else {
128       if (sz > (int) fb.size()) {
129         fb.resize(sz);
130       }
131       for (int i = 0; i < (int) b.size(); i++) {
132         int x = (b[i] % m + m) % m;
133         fb[i] = num(x & ((1 << 15) - 1), x >> 15);
134       }
135       fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
136       fft(fb, sz);
137     }
138     dbl ratio = 0.25 / sz;
139     num r2(0, -1);
140     num r3(ratio, 0);
141     num r4(0, -ratio);
142     num r5(0, 1);
143     for (int i = 0; i <= (sz >> 1); i++) {
144       int j = (sz - i) & (sz - 1);
145       num a1 = (fa[i] + conj(fa[j]));
146       num a2 = (fa[i] - conj(fa[j])) * r2;
147       num b1 = (fb[i] + conj(fb[j])) * r3;
148       num b2 = (fb[i] - conj(fb[j])) * r4;
149       if (i != j) {
150         num c1 = (fa[j] + conj(fa[i]));
151         num c2 = (fa[j] - conj(fa[i])) * r2;
152         num d1 = (fb[j] + conj(fb[i])) * r3;
153         num d2 = (fb[j] - conj(fb[i])) * r4;
154         fa[i] = c1 * d1 + c2 * d2 * r5;
155         fb[i] = c1 * d2 + c2 * d1;
156       }
157       fa[j] = a1 * b1 + a2 * b2 * r5;
158       fb[j] = a1 * b2 + a2 * b1;
159     }
160     fft(fa, sz);
161     fft(fb, sz);
162     vector<int> res(need);
163     for (int i = 0; i < need; i++) {
164       long long aa = fa[i].x + 0.5;
165       long long bb = fb[i].x + 0.5;
166       long long cc = fa[i].y + 0.5;
167       res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
168     }
169     return res;
170   }
171
172   vector<int> square_mod(vector<int> &a, int m) {
173     return multiply_mod(a, a, m, 1);
174   }
175   // fft::multiply uses dbl, outputs vector<long long> of rounded values
176   // fft::multiply_mod might work for res.size() up to 2~21
177   // typedef long double dbl;          =>          up to 2~25 (but takes a lot
          of memory)
178 };
```

6.7 fftfast.cpp

```cpp
1   // FFT_MAXN = 2^k
2   // fft_init() to precalc FFT_MAXN-th roots
3
4   typedef long double db;
5   const int FFT_MAXN = 262144;
6   const int N = 3.1e5;
7   const db pi = acosl(-1.);
8   struct cp {
9       db a, b;
10      cp operator+(const cp &y) const { return (cp){a + y.a, b + y.b}; }
11      cp operator-(const cp &y) const { return (cp){a - y.a, b - y.b}; }
12      cp operator*(const cp &y) const { return (cp){a * y.a - b * y.b, a * y.b
                + b * y.a}; }
13      cp operator!() const { return (cp){a, -b}; };
14  } nw[FFT_MAXN + 1];
15  int bitrev[FFT_MAXN];
16  void dft(cp *a, int n, int flag = 1) {
17      int d = 0;
18      while ((1 << d) * n != FFT_MAXN) d++;
19      rep(i, 0, n - 1) if (i < (bitrev[i] >> d)) swap(a[i], a[bitrev[i] >> d])
                ;
20      for (int l = 2; l <= n; l <<= 1) {
21          int del = FFT_MAXN / l * flag;
22          for (int i = 0; i < n; i += l) {
23              cp *le = a + i, *ri = a + i + (l >> 1), *w = flag == 1 ? nw : nw
                    + FFT_MAXN;
24              rep(k, 0, l / 2 - 1) {
25                  cp ne = *ri * *w;
26                  *ri = *le - ne, *le = *le + ne;
27                  le++, ri++, w += del;
28              }
29          }
30      }
31      if (flag != 1) rep(i, 0, n - 1) a[i].a /= n, a[i].b /= n;
32  }
33  void fft_init() {
34      int L = 0;
35      while ((1 << L) != FFT_MAXN) L++;
36      bitrev[0] = 0;
37      rep(i, 1, FFT_MAXN - 1) bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (L
                - 1));
38      nw[0] = nw[FFT_MAXN] = (cp){1, 0};
39      rep(i, 0, FFT_MAXN)
40      nw[i] = (cp){cosl(2 * pi / FFT_MAXN * i), sinl(2 * pi / FFT_MAXN * i)};
                // very slow
41  }
42
43  void convo(db *a, int n, db *b, int m, db *c) {
44      static cp f[FFT_MAXN >> 1], g[FFT_MAXN >> 1], t[FFT_MAXN >> 1];
45      int N = 2;
46      while (N <= n + m) N <<= 1;
47      rep(i, 0, N - 1) if (i & 1) {
48          f[i >> 1].b = (i <= n) ? a[i] : 0.0;
49          g[i >> 1].b = (i <= m) ? b[i] : 0.0;
50      }
51      else {
52          f[i >> 1].a = (i <= n) ? a[i] : 0.0;
53          g[i >> 1].a = (i <= m) ? b[i] : 0.0;
54      }
55      dft(f, N >> 1);
56      dft(g, N >> 1);
57      int del = FFT_MAXN / (N >> 1);
58      cp qua = (cp){0, 0.25}, one = (cp){1, 0}, four = (cp){4, 0}, *w = nw;
59      rep(i, 0, N / 2 - 1) {
60          int j = i ? (N >> 1) - i : 0;
61          t[i] = (four * !(f[j] * g[j]) - (!f[j] - f[i]) * (!g[j] - g[i]) * (
                one + *w)) * qua;
62          w += del;
63      }
64      dft(t, N >> 1, -1);
65      rep(i, 0, n + m) c[i] = (i & 1) ? t[i >> 1].a : t[i >> 1].b;
66  }
67
68  int mo;
69  void mul(int *a, int *b, int n) {   // n<=N, 0<=a[i],b[i]<mo
70      static cp f[N], g[N], t[N], r[N];
71      int nn = 2;
72      while (nn <= n + n) nn <<= 1;
73      rep(i, 0, nn - 1) {
74          f[i] = (i <= n) ? (cp){(db)(a[i] >> 15), (db)(a[i] & 32767)} : (cp)
                {0, 0};
75          g[i] = (i <= n) ? (cp){(db)(b[i] >> 15), (db)(b[i] & 32767)} : (cp)
                {0, 0};
76      }
77      swap(n, nn);
```

```
78        dft(f, n, 1);
79        dft(g, n, 1);
80        rep(i, 0, n - 1) {
81            int j = i ? n - i : 0;
82            t[i] = ((f[i] + !f[j]) * (!g[j] - g[i]) + (!f[j] - f[i]) * (g[i] + !
                 g[j])) * (cp){0, 0.25};
83            r[i] = (!f[j] - f[i]) * (!g[j] - g[i]) * (cp){-0.25, 0} + (cp){0,
                 0.25} * (f[i] + !f[j]) * (g[i] + !g[j]);
84        }
85        dft(t, n, -1);
86        dft(r, n, -1);
87        rep(i, 0, n - 1)
88        a[i] = ((ll(t[i].a + 0.5) % mo << 15) + ll(r[i].a + 0.5) + (ll(r[i].b +
                 0.5) % mo << 30)) % mo;
89    }
```

6.8   fftnew.cpp

```
1  namespace fft {
2
3  typedef double dbl;
4
5  struct num {
6    dbl x, y;
7    num() { x = y = 0; }
8    num(dbl x_, dbl y_) : x(x_), y(y_) {}
9  };
10
11 inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
12 inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
13 inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a.x *
        b.y + a.y * b.x); }
14 inline num conj(num a) { return num(a.x, -a.y); }
15
16 int base = 1;
17 vector<num> roots = {{0, 0}, {1, 0}};
18 vector<int> rev = {0, 1};
19
20 const dbl PI = static_cast<dbl>(acosl(-1.0));
21
22 void ensure_base(int nbase) {
23   if (nbase <= base) {
24     return;
25   }
26   rev.resize(1 << nbase);
27   for (int i = 0; i < (1 << nbase); i++) {
28     rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
29   }
30   roots.resize(1 << nbase);
31   while (base < nbase) {
32     dbl angle = 2 * PI / (1 << (base + 1));
33 //      num z(cos(angle), sin(angle));
34     for (int i = 1 << (base - 1); i < (1 << base); i++) {
35       roots[i << 1] = roots[i];
36 //        roots[(i << 1) + 1] = roots[i] * z;
37       dbl angle_i = angle * (2 * i + 1 - (1 << base));
38       roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
39     }
40     base++;
41   }
42 }
43
44 void fft(vector<num>& a, int n = -1) {
45   if (n == -1) {
46     n = (int) a.size();
47   }
48   assert((n & (n - 1)) == 0);
49   int zeros = __builtin_ctz(n);
50   ensure_base(zeros);
51   int shift = base - zeros;
52   for (int i = 0; i < n; i++) {
53     if (i < (rev[i] >> shift)) {
54       swap(a[i], a[rev[i] >> shift]);
55     }
56   }
57   for (int k = 1; k < n; k <<= 1) {
58     for (int i = 0; i < n; i += 2 * k) {
59       for (int j = 0; j < k; j++) {
60         num z = a[i + j + k] * roots[j + k];
61         a[i + j + k] = a[i + j] - z;
62         a[i + j] = a[i + j] + z;
63       }
64     }
65   }
66 }
67
```

```
 68   vector<num> fa, fb;
 69
 70   vector<int64_t> square(const vector<int>& a) {
 71     if (a.empty()) {
 72       return {};
 73     }
 74     int need = (int) a.size() + (int) a.size() - 1;
 75     int nbase = 1;
 76     while ((1 << nbase) < need) nbase++;
 77     ensure_base(nbase);
 78     int sz = 1 << nbase;
 79     if ((sz >> 1) > (int) fa.size()) {
 80       fa.resize(sz >> 1);
 81     }
 82     for (int i = 0; i < (sz >> 1); i++) {
 83       int x = (2 * i < (int) a.size() ? a[2 * i] : 0);
 84       int y = (2 * i + 1 < (int) a.size() ? a[2 * i + 1] : 0);
 85       fa[i] = num(x, y);
 86     }
 87     fft(fa, sz >> 1);
 88     num r(1.0 / (sz >> 1), 0.0);
 89     for (int i = 0; i <= (sz >> 2); i++) {
 90       int j = ((sz >> 1) - i) & ((sz >> 1) - 1);
 91       num fe = (fa[i] + conj(fa[j])) * num(0.5, 0);
 92       num fo = (fa[i] - conj(fa[j])) * num(0, -0.5);
 93       num aux = fe * fe + fo * fo * roots[(sz >> 1) + i] * roots[(sz >> 1) + i
            ];
 94       num tmp = fe * fo;
 95       fa[i] = r * (conj(aux) + num(0, 2) * conj(tmp));
 96       fa[j] = r * (aux + num(0, 2) * tmp);
 97     }
 98     fft(fa, sz >> 1);
 99     vector<int64_t> res(need);
100     for (int i = 0; i < need; i++) {
101       res[i] = llround(i % 2 == 0 ? fa[i >> 1].x : fa[i >> 1].y);
102     }
103     return res;
104   }
105
106   vector<int64_t> multiply(const vector<int>& a, const vector<int>& b) {
107     if (a.empty() || b.empty()) {
108       return {};
109     }
110     if (a == b) {
111       return square(a);
112     }
113     int need = (int) a.size() + (int) b.size() - 1;
114     int nbase = 1;
115     while ((1 << nbase) < need) nbase++;
116     ensure_base(nbase);
117     int sz = 1 << nbase;
118     if (sz > (int) fa.size()) {
119       fa.resize(sz);
120     }
121     for (int i = 0; i < sz; i++) {
122       int x = (i < (int) a.size() ? a[i] : 0);
123       int y = (i < (int) b.size() ? b[i] : 0);
124       fa[i] = num(x, y);
125     }
126     fft(fa, sz);
127     num r(0, -0.25 / (sz >> 1));
128     for (int i = 0; i <= (sz >> 1); i++) {
129       int j = (sz - i) & (sz - 1);
130       num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
131       fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
132       fa[i] = z;
133     }
134     for (int i = 0; i < (sz >> 1); i++) {
135       num A0 = (fa[i] + fa[i + (sz >> 1)]) * num(0.5, 0);
136       num A1 = (fa[i] - fa[i + (sz >> 1)]) * num(0.5, 0) * roots[(sz >> 1) + i
            ];
137       fa[i] = A0 + A1 * num(0, 1);
138     }
139     fft(fa, sz >> 1);
140     vector<int64_t> res(need);
141     for (int i = 0; i < need; i++) {
142       res[i] = llround(i % 2 == 0 ? fa[i >> 1].x : fa[i >> 1].y);
143     }
144     return res;
145   }
146
147   vector<int> multiply_mod(const vector<int>& a, const vector<int>& b, int m)
          {
148     if (a.empty() || b.empty()) {
149       return {};
150     }
```

```
151    int eq = (a.size() == b.size() && a == b);
152    int need = (int) a.size() + (int) b.size() - 1;
153    int nbase = 0;
154    while ((1 << nbase) < need) nbase++;
155    ensure_base(nbase);
156    int sz = 1 << nbase;
157    if (sz > (int) fa.size()) {
158      fa.resize(sz);
159    }
160    for (int i = 0; i < (int) a.size(); i++) {
161      int x = (a[i] % m + m) % m;
162      fa[i] = num(x & ((1 << 15) - 1), x >> 15);
163    }
164    fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
165    fft(fa, sz);
166    if (sz > (int) fb.size()) {
167      fb.resize(sz);
168    }
169    if (eq) {
170      copy(fa.begin(), fa.begin() + sz, fb.begin());
171    } else {
172      for (int i = 0; i < (int) b.size(); i++) {
173        int x = (b[i] % m + m) % m;
174        fb[i] = num(x & ((1 << 15) - 1), x >> 15);
175      }
176      fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
177      fft(fb, sz);
178    }
179    dbl ratio = 0.25 / sz;
180    num r2(0, -1);
181    num r3(ratio, 0);
182    num r4(0, -ratio);
183    num r5(0, 1);
184    for (int i = 0; i <= (sz >> 1); i++) {
185      int j = (sz - i) & (sz - 1);
186      num a1 = (fa[i] + conj(fa[j]));
187      num a2 = (fa[i] - conj(fa[j])) * r2;
188      num b1 = (fb[i] + conj(fb[j])) * r3;
189      num b2 = (fb[i] - conj(fb[j])) * r4;
190      if (i != j) {
191        num c1 = (fa[j] + conj(fa[i]));
192        num c2 = (fa[j] - conj(fa[i])) * r2;
193        num d1 = (fb[j] + conj(fb[i])) * r3;
194        num d2 = (fb[j] - conj(fb[i])) * r4;
195        fa[i] = c1 * d1 + c2 * d2 * r5;
196        fb[i] = c1 * d2 + c2 * d1;
197      }
198      fa[j] = a1 * b1 + a2 * b2 * r5;
199      fb[j] = a1 * b2 + a2 * b1;
200    }
201    fft(fa, sz);
202    fft(fb, sz);
203    vector<int> res(need);
204    for (int i = 0; i < need; i++) {
205      int64_t aa = llround(fa[i].x);
206      int64_t bb = llround(fb[i].x);
207      int64_t cc = llround(fa[i].y);
208      res[i] = static_cast<int>((aa + ((bb % m) << 15) + ((cc % m) << 30)) % m
           );
209    }
210    return res;
211  }
212
213  } // namespace fft
```

6.9   FST.cpp

```
1  void fst(VI &a,bool inv) {
2      for (int n=SZ(a),step=1;step<n;step*=2) {
3          for (int i=0;i<n;i+=2*step) rep(j,i,i+step-1) {
4              int &u=a[j],&v=a[j+step];
5              tie(u,v)=
6              inv?PII(v-u,u):PII(v,u+v); // AND
7              inv?PII(v,u-v):PII(u+v,u); // OR
8              PII(u+v,u-v); // XOR
9          }
10     }
11     if (inv) for (auto &x : a) x/=SZ(a); // XOR only
12 }
13 VI conv(VI a,VI b) {
14     fst(a,0),fst(b,0);
15     rep(i,0,SZ(a)-1) a[i]=a[i]*b[i];
16     fst(a,1); return a;
17 }
```

6.10 FWT.cpp

```
1   ll f[maxn], g[maxn], h[maxn];
2   int main() {
3       for (int i = 0; i < n; i++) {
4           for (int j = 0; j < bit(n); j++) {
5               if ((j & bit(i)) == 0) {
6                   f[j] += f[j + bit(i)];
7                   g[j] += g[j + bit(i)];
8               }
9           }
10      }
11      for (int i = 0; i < bit(n); i++) {
12          f[i] %= mod;
13          g[i] %= mod;
14          h[i] = f[i] * g[i] % mod;
15      }
16      for (int i = 0; i < n; i++) {
17          for (int j = 0; j < bit(n); j++) {
18              if ((j & bit(i)) == 0)
19                  h[j] -= h[j + bit(i)];
20          }
21      }
22      for (int i = 0; i < bit(n); i++) {
23          h[i] %= mod;
24          if (h[i] < 0) h[i] += mod;
25      }
26
27      ll ans = 0;
28      rep(i, 0, bit(n) - 1) ans ^= h[i];
29      cout << ans << '\n';
30  }
```

6.11 gauss(合数).cpp

```
1   void gauss(int n) {
2       int ans = 1;
3       //rep(i,1,n) rep(j,1,n) p[i][j]%=mod;
4       for (int i = 1; i <= n; i++) {
5           for (int j = i + 1; j <= n; j++) {
6               int x = i, y = j;
7               while (p[x][i]) {
8                   int t = p[y][i] / p[x][i];
9                   for (int k = i; k <= n; k++)
```

```
10                      p[y][k] = (p[y][k] - p[x][k] * t) % mod;
11                  swap(x, y);
12              }
13              if (x == i) {
14                  for (int k = i; k <= n; k++) swap(p[i][k], p[j][k]);
15                  ans = -ans;
16              }
17          }
18      }
19  }
```

6.12 gauss.cpp

```
1   ll f[N][N];
2   ll v[N], a[N];
3   void gauss() {
4       for (int i = 1; i <= n; i++) {
5           for (int j = i; j <= n; j++) {
6               if (f[j][i] > f[i][i]) {
7                   swap(v[i], v[j]);
8                   for (int k = 1; k <= n; k++)
9                       swap(f[j][k], f[i][k]);
10              }
11          }
12          for (int j = i + 1; j <= n; j++) {
13              if (f[j][i]) {
14                  int delta = f[j][i] * fpow(f[i][i], mod - 2) % mod;
15                  for (int k = i; k <= n; k++) {
16                      f[j][k] -= f[i][k] * delta % mod;
17                      if (f[j][k] < 0)
18                          f[j][k] += mod;
19                  }
20                  v[j] -= v[i] * delta % mod;
21                  if (v[j] < mod)
22                      v[j] += mod;
23              }
24          }
25      }
26      for (int j = n; j > 0; j--) {
27          for (int k = j + 1; k <= n; k++) {
28              v[j] -= f[j][k] * a[k] % mod;
29              if (v[j] < 0)
30                  v[j] += mod;
```

```
31                }
32            a[j] = v[j] * fpow(f[j][j], mod - 2) % mod;
33        }
34  }
```

6.13 linearbasis.cpp

```
1   struct linear_base {
2       ll w[64];
3       ll zero = 0;
4       ll tot = -1;
5       void clear() {
6           rep(i, 0, 63) w[i] = 0;
7           zero = 0;
8           tot = -1;
9       }
10      void insert(ll x) {
11          for (int i = 62; i >= 0; i--) {
12              if (x & bit(i))
13                  if (!w[i]) {w[i] = x; return;}
14                  else x ^= w[i];
15          }
16          zero++;
17      }
18      void build() {
19          rep(i, 0, 63) rep(j, 0, i - 1) {
20              if (w[i]&bit(j)) w[i] ^= w[j];
21          }
22          for (int i = 0; i <= 62; i++) {
23              if (w[i] != 0) w[++tot] = w[i];
24          }
25      }
26      ll qmax() {
27          ll res = 0;
28          for (int i = 62; i >= 0; i--) {
29              res = max(res, res ^ w[i]);
30          }
31          return res;
32      }
33      bool check(ll x) {
34          for (int i = 62; i >= 0; i--) {
35              if (x & bit(i))
36                  if (!w[i]) return false;
37                  else x ^= w[i];
38          }
39          return true;
40      }
41      ll query(ll k) {
42          ll res = 0;
43          // if (zero) k-=1;
44          // if (k >= bit(tot)) return -1;
45          for (int i = tot; i >= 0; i--) {
46              if (k & bit(i)) {
47                  res = max(res, res ^ w[i]);
48              } else {
49                  res = min(res, res ^ w[i]);
50              }
51          }
52          return res;
53      }
54  };
```

6.14 lucas.cpp

```
1   ll fac[maxn], fnv[maxn];
2
3   ll binom(ll a, ll b) {
4       if (b > a || b < 0) return 0;
5       return fac[a] * fnv[a - b] % p * fnv[b] % p;
6   }
7
8   ll lucas(ll a, ll b, ll p) {
9       ll ans = 1;
10      while (a > 0 || b > 0) {
11          ans = (ans * binom(a % p, b % p)) % p;
12          a /= p, b /= p;
13      }
14      return ans;
15  }
16
17  int main() {
18      cin >> p >> T;
19      fac[0] = 1;
20      rep(i, 1, p - 1) fac[i] = fac[i - 1] * i % p;
21      fnv[p - 1] = powmod(fac[p - 1], p - 2, p);
22      per(i, p - 2, 0) fnv[i] = fnv[i + 1] * (i + 1) % p;
```

```
23        assert(fnv[0] == 1);
24  }
```

6.15   mathdiv.cpp

```
1   ll floor_div(ll x, ll y) {
2       assert(y != 0);
3       if (y < 0) {
4           y = -y;
5           x = -x;
6       }
7       if (x >= 0) return x / y;
8       return (x + 1) / y - 1;
9   }
10  ll ceil_div(ll x, ll y) {
11      assert(y != 0);
12      if (y < 0) {
13          y = -y;
14          x = -x;
15      }
16      if (x <= 0) return x / y;
17      return (x - 1) / y + 1;
18  }
```

6.16   matrix.cpp

```
1   template <typename T>
2   vector<vector<T>> operator*(const vector<vector<T>>& a, const vector<vector<
        T>>& b) {
3     if (a.empty() || b.empty()) {
4       return {{}};
5     }
6     vector<vector<T>> c(a.size(), vector<T>(b[0].size()));
7     for (int i = 0; i < static_cast<int>(c.size()); i++) {
8       for (int j = 0; j < static_cast<int>(c[0].size()); j++) {
9         c[i][j] = 0;
10        for (int k = 0; k < static_cast<int>(b.size()); k++) {
11          c[i][j] += a[i][k] * b[k][j];
12        }
13      }
14    }
15    return c;
16  }
```

```
17
18  template <typename T>
19  vector<vector<T>>& operator*=(vector<vector<T>>& a, const vector<vector<T>>&
        b) {
20    return a = a * b;
21  }
22
23  template <typename T, typename U>
24  vector<vector<T>> power(const vector<vector<T>>& a, const U& b) {
25    assert(b >= 0);
26    vector<U> binary;
27    U bb = b;
28    while (bb > 0) {
29      binary.push_back(bb & 1);
30      bb >>= 1;
31    }
32    vector<vector<T>> res(a.size(), vector<T>(a.size()));
33    for (int i = 0; i < static_cast<int>(a.size()); i++) {
34      res[i][i] = 1;
35    }
36    for (int j = (int)binary.size() - 1; j >= 0; j--) {
37      res *= res;
38      if (binary[j] == 1) {
39        res *= a;
40      }
41    }
42    return res;
43  }
```

6.17   matrixfast.cpp

```
1   Description: Basic operations on square matrices.
2   Usage: Matrix<int, 3> A;
3   A.d = {{{{1, 2, 3}}, {{4, 5, 6}}, {{7, 8, 9}}}};
4   vector<int> vec = {1, 2, 3};
5   vec = (A^N) * vec;
6
7   template<class T, int N> struct Matrix {
8       typedef Matrix M;
9       array<array<T, N>, N> d{};
10      M operator*(const M& m) const {
11          M a;
12          rep(i, 0, N) rep(j, 0, N)
```

```
13        rep(k, 0, N) a.d[i][j] += d[i][k] * m.d[k][j];
14        return a;
15    }
16    vector<T> operator*(const vector<T>& vec) const {
17        vector<T> ret(N);
18        rep(i, 0, N) rep(j, 0, N) ret[i] += d[i][j] * vec[j];
19        return ret;
20    }
21    M operator^(ll p) const {
22        assert(p >= 0);
23        M a, b(*this);
24        rep(i, 0, N) a.d[i][i] = 1;
25        while (p) {
26            if (p & 1) a = a * b;
27            b = b * b;
28            p >>= 1;
29        }
30        return a;
31    }
32 };
```

6.18   MillerRabbin pollard modmul.cpp

```
1  /*ModMulLL.h
2  Description: Calculate a·b mod c (or a
3  b mod c) for 0   a, b   c   7.2·10^18
4  Time: O (1) for modmul, O (log b) for modpow*/
5  /*ull modmul(ull a, ull b, ull M) {
6      ll ret = a * b - M * ull(1.L / M * a * b);
7      return ret + M * (ret < 0) - M * (ret >= (ll)M);
8  }
9  ull modpow(ull b, ull e, ull mod) {
10     ull ans = 1;
11     for (; e; b = modmul(b, b, mod), e /= 2)
12         if (e & 1) ans = modmul(ans, b, mod);
13     return ans;
14 }*/
15 ll modmul(ll a, ll b, ll m) {
16     a %= m, b %= m;
17     ll d = ((ldb)a * b / m);
18     d = a * b - d * m;
19     if (d >= m) d -= m;
20     if (d < 0) d += m;
```

```
21     return d;
22 }
23 ll modpow(ll a, ll b, ll p) {
24     ll ans = 1;
25     while (b) {
26         if (b & 1) ans = modmul(ans, a, p);
27         a = modmul(a, a, p); b >>= 1;
28     } return ans;
29 }
30 /*MillerRabin.h
31 Description: Deterministic Miller-Rabin primality test. Guaranteed to
32 work for numbers up to 7 · 1018; for larger numbers, use Python and extend A
        randomly.
33 Time: 7 times the complexity of a^b mod c.*/
34 bool isPrime(ll n) {
35     if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
36     ll A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
37            s = __builtin_ctzll(n - 1), d = n >> s;
38     for (ll a : A) { // ^ count trailing zeroes
39         ll p = modpow(a % n, d, n), i = s;
40         while (p != 1 && p != n - 1 && a % n && i--)
41             p = modmul(p, p, n);
42         if (p != n - 1 && i != s) return 0;
43     }
44     return 1;
45 }
46 /*Factor.h
47 Description: Pollard-rho randomized factorization algorithm. Returns
48 prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
49 Time: O(n^1/4), less for numbers with small factors.*/
50 ll pollard(ll n) {
51     auto f = [n](ll x) { return modmul(x, x, n) + 1; };
52     ll x = 0, y = 0, t = 30, prd = 2, i = 1, q;
53     while (t++ % 40 || __gcd(prd, n) == 1) {
54         if (x == y) x = ++i, y = f(x);
55         if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
56         x = f(x), y = f(f(y));
57     }
58     return __gcd(prd, n);
59 }
60 vector<ll> factor(ll n) {
61     if (n == 1) return {};
62     if (isPrime(n)) return {n};
```

```
63      ll x = pollard(n);
64      auto l = factor(x), r = factor(n / x);
65      l.insert(l.end(), all(r));
66      return l;
67  }
```

6.19   ntt(polynomial).cpp

```
1   #include<bits/stdc++.h>
2   using namespace std;
3
4   const int mod = 998244353;
5
6   inline void add(int &x, int y) {
7     x += y;
8     if (x >= mod) {
9       x -= mod;
10    }
11  }
12
13  inline void sub(int &x, int y) {
14    x -= y;
15    if (x < 0) {
16      x += mod;
17    }
18  }
19
20  inline int mul(int x, int y) {
21    return (long long) x * y % mod;
22  }
23
24  inline int power(int x, int y) {
25    int res = 1;
26    for (; y; y >>= 1, x = mul(x, x)) {
27      if (y & 1) {
28        res = mul(res, x);
29      }
30    }
31    return res;
32  }
33
34  inline int inv(int a) {
35    a %= mod;
```

```
36    if (a < 0) {
37      a += mod;
38    }
39    int b = mod, u = 0, v = 1;
40    while (a) {
41      int t = b / a;
42      b -= t * a;
43      swap(a, b);
44      u -= t * v;
45      swap(u, v);
46    }
47    if (u < 0) {
48      u += mod;
49    }
50    return u;
51  }
52
53  namespace ntt {
54  int base = 1, root = -1, max_base = -1;
55  vector<int> rev = {0, 1}, roots = {0, 1};
56
57  void init() {
58    int temp = mod - 1;
59    max_base = 0;
60    while (temp % 2 == 0) {
61      temp >>= 1;
62      ++max_base;
63    }
64    root = 2;
65    while (true) {
66      if (power(root, 1 << max_base) == 1 && power(root, 1 << (max_base - 1))
            != 1) {
67        break;
68      }
69      ++root;
70    }
71  }
72
73  void ensure_base(int nbase) {
74    if (max_base == -1) {
75      init();
76    }
77    if (nbase <= base) {
```

72

```cpp
 78        return;
 79      }
 80      assert(nbase <= max_base);
 81      rev.resize(1 << nbase);
 82      for (int i = 0; i < 1 << nbase; ++i) {
 83        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (nbase - 1));
 84      }
 85      roots.resize(1 << nbase);
 86      while (base < nbase) {
 87        int z = power(root, 1 << (max_base - 1 - base));
 88        for (int i = 1 << (base - 1); i < 1 << base; ++i) {
 89          roots[i << 1] = roots[i];
 90          roots[i << 1 | 1] = mul(roots[i], z);
 91        }
 92        ++base;
 93      }
 94    }
 95
 96    void dft(vector<int> &a) {
 97      int n = a.size(), zeros = __builtin_ctz(n);
 98      ensure_base(zeros);
 99      int shift = base - zeros;
100      for (int i = 0; i < n; ++i) {
101        if (i < rev[i] >> shift) {
102          swap(a[i], a[rev[i] >> shift]);
103        }
104      }
105      for (int i = 1; i < n; i <<= 1) {
106        for (int j = 0; j < n; j += i << 1) {
107          for (int k = 0; k < i; ++k) {
108            int x = a[j + k], y = mul(a[j + k + i], roots[i + k]);
109            a[j + k] = (x + y) % mod;
110            a[j + k + i] = (x + mod - y) % mod;
111          }
112        }
113      }
114    }
115
116    vector<int> multiply(vector<int> a, vector<int> b) {
117      int need = a.size() + b.size() - 1, nbase = 0;
118      while (1 << nbase < need) {
119        ++nbase;
120      }
121      ensure_base(nbase);
122      int sz = 1 << nbase;
123      a.resize(sz);
124      b.resize(sz);
125      bool equal = a == b;
126      dft(a);
127      if (equal) {
128        b = a;
129      } else {
130        dft(b);
131      }
132      int inv_sz = inv(sz);
133      for (int i = 0; i < sz; ++i) {
134        a[i] = mul(mul(a[i], b[i]), inv_sz);
135      }
136      reverse(a.begin() + 1, a.end());
137      dft(a);
138      a.resize(need);
139      return a;
140    }
141
142    vector<int> inverse_new(const vector<int> &a) {
143      assert(!a.empty());
144      int n = (int) a.size();
145      vector<int> b = {inv(a[0])};
146      while ((int) b.size() < n) {
147        vector<int> x(a.begin(), a.begin() + min(a.size(), b.size() << 1));
148        x.resize(b.size() << 1);
149        b.resize(b.size() << 1);
150        vector<int> c = b;
151        // NTT<T>::fft(c);
152        // NTT<T>::fft(x);
153        dft(c);
154        dft(x);
155        // Modular<T> inv = 1 / static_cast<Modular<T>>((int) x.size());
156        int inv_sz = inv((int)x.size());
157        for (int i = 0; i < (int) x.size(); i++) {
158          // x[i] *= c[i] * inv;
159          x[i] = mul(x[i], mul(c[i], inv_sz));
160        }
161        reverse(x.begin() + 1, x.end());
162        // NTT<T>::fft(x);
163        dft(x);
```

```
164      rotate(x.begin(), x.begin() + (x.size() >> 1), x.end());
165      fill(x.begin() + (x.size() >> 1), x.end(), 0);
166      // NTT<T>::fft(x);
167      dft(x);
168      for (int i = 0; i < (int) x.size(); i++) {
169        // x[i] *= c[i] * inv;
170        x[i] = mul(x[i], mul(c[i], inv_sz));
171      }
172      reverse(x.begin() + 1, x.end());
173      // NTT<T>::fft(x);
174      dft(x);
175      for (int i = 0; i < ((int) x.size() >> 1); i++) {
176        // b[i + ((int) x.size() >> 1)] = -x[i];
177        int t = 0; sub(t, x[i]);
178        b[i + ((int) x.size() >> 1)] = t;
179      }
180    }
181    b.resize(n);
182    return b;
183  }
184
185  vector<int> inverse(vector<int> a) {
186    int n = a.size(), m = (n + 1) >> 1;
187    if (n == 1) {
188      return vector<int>(1, inv(a[0]));
189    } else {
190      vector<int> b = inverse(vector<int>(a.begin(), a.begin() + m));
191      int need = n << 1, nbase = 0;
192      while (1 << nbase < need) {
193        ++nbase;
194      }
195      ensure_base(nbase);
196      int sz = 1 << nbase;
197      a.resize(sz);
198      b.resize(sz);
199      dft(a);
200      dft(b);
201      int inv_sz = inv(sz);
202      for (int i = 0; i < sz; ++i) {
203        a[i] = mul(mul(mod + 2 - mul(a[i], b[i]), b[i]), inv_sz);
204      }
205      reverse(a.begin() + 1, a.end());
206      dft(a);
207      a.resize(n);
208      return a;
209    }
210  }
211  }
212
213  using ntt::multiply;
214  using ntt::inverse;
215
216  vector<int>& operator += (vector<int> &a, const vector<int> &b) {
217    if (a.size() < b.size()) {
218      a.resize(b.size());
219    }
220    for (int i = 0; i < b.size(); ++i) {
221      add(a[i], b[i]);
222    }
223    return a;
224  }
225
226  vector<int> operator + (const vector<int> &a, const vector<int> &b) {
227    vector<int> c = a;
228    return c += b;
229  }
230
231  vector<int>& operator -= (vector<int> &a, const vector<int> &b) {
232    if (a.size() < b.size()) {
233      a.resize(b.size());
234    }
235    for (int i = 0; i < b.size(); ++i) {
236      sub(a[i], b[i]);
237    }
238    return a;
239  }
240
241  vector<int> operator - (const vector<int> &a, const vector<int> &b) {
242    vector<int> c = a;
243    return c -= b;
244  }
245
246  vector<int>& operator *= (vector<int> &a, const vector<int> &b) {
247    if (min(a.size(), b.size()) < 128) {
248      vector<int> c = a;
249      a.assign(a.size() + b.size() - 1, 0);
```

```cpp
250      for (int i = 0; i < c.size(); ++i) {
251        for (int j = 0; j < b.size(); ++j) {
252          add(a[i + j], mul(c[i], b[j]));
253        }
254      }
255    } else {
256      a = multiply(a, b);
257    }
258    return a;
259  }
260
261  vector<int> operator * (const vector<int> &a, const vector<int> &b) {
262    vector<int> c = a;
263    return c *= b;
264  }
265
266  vector<int>& operator /= (vector<int> &a, const vector<int> &b) {
267    int n = a.size(), m = b.size();
268    if (n < m) {
269      a.clear();
270    } else {
271      vector<int> c = b;
272      reverse(a.begin(), a.end());
273      reverse(c.begin(), c.end());
274      c.resize(n - m + 1);
275      a *= inverse(c);
276      a.erase(a.begin() + n - m + 1, a.end());
277      reverse(a.begin(), a.end());
278    }
279    return a;
280  }
281
282  vector<int> operator / (const vector<int> &a, const vector<int> &b) {
283    vector<int> c = a;
284    return c /= b;
285  }
286
287  vector<int>& operator %= (vector<int> &a, const vector<int> &b) {
288    int n = a.size(), m = b.size();
289    if (n >= m) {
290      vector<int> c = (a / b) * b;
291      a.resize(m - 1);
292      for (int i = 0; i < m - 1; ++i) {
293        sub(a[i], c[i]);
294      }
295    }
296    return a;
297  }
298
299  vector<int> operator % (const vector<int> &a, const vector<int> &b) {
300    vector<int> c = a;
301    return c %= b;
302  }
303
304  vector<int> derivative(const vector<int> &a) {
305    int n = a.size();
306    vector<int> b(n - 1);
307    for (int i = 1; i < n; ++i) {
308      b[i - 1] = mul(a[i], i);
309    }
310    return b;
311  }
312
313  vector<int> primitive(const vector<int> &a) {
314    int n = a.size();
315    vector<int> b(n + 1), invs(n + 1);
316    for (int i = 1; i <= n; ++i) {
317      invs[i] = i == 1 ? 1 : mul(mod - mod / i, invs[mod % i]);
318      b[i] = mul(a[i - 1], invs[i]);
319    }
320    return b;
321  }
322
323  vector<int> logarithm(const vector<int> &a) {
324    vector<int> b = primitive(derivative(a) * inverse(a));
325    b.resize(a.size());
326    return b;
327  }
328
329  vector<int> exponent(const vector<int> &a) {
330    vector<int> b(1, 1);
331    while (b.size() < a.size()) {
332      vector<int> c(a.begin(), a.begin() + min(a.size(), b.size() << 1));
333      add(c[0], 1);
334      vector<int> old_b = b;
335      b.resize(b.size() << 1);
```

```
336      c -= logarithm(b);
337      c *= old_b;
338      for (int i = b.size() >> 1; i < b.size(); ++i) {
339        b[i] = c[i];
340      }
341    }
342    b.resize(a.size());
343    return b;
344  }
345
346  vector<int> power(vector<int> a, int m) {
347    int n = a.size(), p = -1;
348    vector<int> b(n);
349    for (int i = 0; i < n; ++i) {
350      if (a[i]) {
351        p = i;
352        break;
353      }
354    }
355    if (p == -1) {
356      b[0] = !m;
357      return b;
358    }
359    if ((long long) m * p >= n) {
360      return b;
361    }
362    int mu = power(a[p], m), di = inv(a[p]);
363    vector<int> c(n - m * p);
364    for (int i = 0; i < n - m * p; ++i) {
365      c[i] = mul(a[i + p], di);
366    }
367    c = logarithm(c);
368    for (int i = 0; i < n - m * p; ++i) {
369      c[i] = mul(c[i], m);
370    }
371    c = exponent(c);
372    for (int i = 0; i < n - m * p; ++i) {
373      b[i + m * p] = mul(c[i], mu);
374    }
375    return b;
376  }
377
378  vector<int> sqrt(const vector<int> &a) {
```

```
379    vector<int> b(1, 1);
380    while (b.size() < a.size()) {
381      vector<int> c(a.begin(), a.begin() + min(a.size(), b.size() << 1));
382      vector<int> old_b = b;
383      b.resize(b.size() << 1);
384      c *= inverse(b);
385      for (int i = b.size() >> 1; i < b.size(); ++i) {
386        b[i] = mul(c[i], (mod + 1) >> 1);
387      }
388    }
389    b.resize(a.size());
390    return b;
391  }
392
393  vector<int> multiply_all(int l, int r, vector<vector<int>> &all) {
394    if (l > r) {
395      return vector<int>();
396    } else if (l == r) {
397      return all[l];
398    } else {
399      int y = (l + r) >> 1;
400      return multiply_all(l, y, all) * multiply_all(y + 1, r, all);
401    }
402  }
403
404  vector<int> evaluate(const vector<int> &f, const vector<int> &x) {
405    int n = x.size();
406    if (!n) {
407      return vector<int>();
408    }
409    vector<vector<int>> up(n * 2);
410    for (int i = 0; i < n; ++i) {
411      up[i + n] = vector<int> {(mod - x[i]) % mod, 1};
412    }
413    for (int i = n - 1; i; --i) {
414      up[i] = up[i << 1] * up[i << 1 | 1];
415    }
416    vector<vector<int>> down(n * 2);
417    down[1] = f % up[1];
418    for (int i = 2; i < n * 2; ++i) {
419      down[i] = down[i >> 1] % up[i];
420    }
421    vector<int> y(n);
```

```
422    for (int i = 0; i < n; ++i) {
423      y[i] = down[i + n][0];
424    }
425    return y;
426  }
427
428  vector<int> interpolate(const vector<int> &x, const vector<int> &y) {
429    int n = x.size();
430    vector<vector<int>> up(n * 2);
431    for (int i = 0; i < n; ++i) {
432      up[i + n] = vector<int> {(mod - x[i]) % mod, 1};
433    }
434    for (int i = n - 1; i; --i) {
435      up[i] = up[i << 1] * up[i << 1 | 1];
436    }
437    vector<int> a = evaluate(derivative(up[1]), x);
438    for (int i = 0; i < n; ++i) {
439      a[i] = mul(y[i], inv(a[i]));
440    }
441    vector<vector<int>> down(n * 2);
442    for (int i = 0; i < n; ++i) {
443      down[i + n] = vector<int>(1, a[i]);
444    }
445    for (int i = n - 1; i; --i) {
446      down[i] = down[i << 1] * up[i << 1 | 1] + down[i << 1 | 1] * up[i << 1];
447    }
448    return down[1];
449  }
450
451  int main() {
452
453  }
```

6.20   simplex.cpp

```
1   /**
2    * Author: Stanford
3    * Source: Stanford Notebook
4    * License: MIT
5    * Description: Solves a general linear maximization problem: maximize $c^T
         x$ subject to $Ax \le b$, $x \ge 0$.
6    * Returns -inf if there is no solution, inf if there are arbitrarily good
         solutions, or the maximum value of $c^T x$ otherwise.
7    * The input vector is set to an optimal $x$ (or in the unbounded case, an
         arbitrary solution fulfilling the constraints).
8    * Numerical stability is not guaranteed. For better performance, define
         variables such that $x = 0$ is viable.
9    * Usage:
10   * vvd A = {{1,-1}, {-1,1}, {-1,-2}};
11   * vd b = {1,1,-4}, c = {-1,-1}, x;
12   * T val = LPSolver(A, b, c).solve(x);
13   * Time: O(NM * \#pivots), where a pivot may be e.g. an edge relaxation. O
         (2^n) in the general case.
14   * Status: seems to work?
15   */
16
17  typedef long double T; // long double, Rational, double + mod<P>...
18  typedef vector<T> vd;
19  typedef vector<vd> vvd;
20
21  const T eps = 1e-8, inf = 1/.0;
22  #define MP make_pair
23  #define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
24
25  struct LPSolver {
26      int m, n;
27      vector<int> N, B;
28      vvd D;
29
30      LPSolver(const vvd& A, const vd& b, const vd& c) :
31          m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, vd(n+2)) {
32              for(int i = 0; i < m; i++){
33                  for(int j = 0; j < n; j++){
34                      D[i][j] = A[i][j];
35                  }
36              }
37              for(int i = 0; i < m; i++){
38                  B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];
39              }
40              for(int j = 0; j < n; j++){
41                  N[j] = j; D[m][j] = -c[j];
42              }
43              N[n] = -1; D[m+1][n] = 1;
44          }
45
46      void pivot(int r, int s) {
```

```
47          T *a = D[r].data(), inv = 1 / a[s];
48          for(int i = 0; i < m+2; i++){
49              if (i != r && abs(D[i][s]) > eps) {
50                  T *b = D[i].data(), inv2 = b[s] * inv;
51                  for(int j = 0; j < n+2; j++){
52                      b[j] -= a[j] * inv2;
53                  }
54                  b[s] = a[s] * inv2;
55              }
56          }
57          for(int j = 0; j < n+2; j++){
58              if (j != s) D[r][j] *= inv;
59          }
60          for(int i = 0; i < m+2; i++){
61              if (i != r) D[i][s] *= -inv;
62          }
63          D[r][s] = inv;
64          swap(B[r], N[s]);
65      }
66
67      bool simplex(int phase) {
68          int x = m + phase - 1;
69          for (;;) {
70              int s = -1;
71              for(int j = 0; j < n+1; j++){
72                  if (N[j] != -phase) ltj(D[x]);
73              }
74              if (D[x][s] >= -eps) return true;
75              int r = -1;
76              for(int i = 0; i < m; i++){
77                  if (D[i][s] <= eps) continue;
78                  if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
79                          < MP(D[r][n+1] / D[r][s], B[r])) r = i;
80              }
81              if (r == -1) return false;
82              pivot(r, s);
83          }
84      }
85
86      T solve(vd &x) {
87          int r = 0;
88          for(int i = 1; i < m; i++){
89              if (D[i][n+1] < D[r][n+1]) r = i;
```

```
90              }
91              if (D[r][n+1] < -eps) {
92                  pivot(r, n);
93                  if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
94                  for(int i = 0; i < m; i++) if (B[i] == -1) {
95                      int s = 0;
96                      for(int j = 1; j < n+1; j++){
97                          ltj(D[i]);
98                      }
99                      pivot(i, s);
100                 }
101             }
102             bool ok = simplex(1); x = vd(n);
103             for(int i = 0; i < m; i++){
104                 if (B[i] < n) x[B[i]] = D[i][n+1];
105             }
106             return ok ? D[m][n+1] : inf;
107         }
108     };
```

6.21 区间互质.cpp

```
1   int p[N / 5], num;
2   void prime(int n) {
3       num = 0;
4       for (int i = 2; i * i <= n; i++) {
5           if ((n % i) == 0) {
6               p[++num] = i;
7               while ((n % i) == 0) n /= i;
8           }
9       }
10      if (n > 1) p[++num] = n;
11  }
12  ll solve(ll r, int k) {
13      prime(k);
14      ll res = 0;
15      for (int i = 1; i < (1 << num); i++) {
16          int k = 0;
17          ll div = 1;
18          for (int j = 1; j <= num; j++) {
19              if (i & (1 << (j - 1))) {
20                  k++;
21                  div *= p[j];
```

```
22                }
23            }
24            if (k % 2)
25                res += r / div;
26            else
27                res -= r / div;
28        }
29        return r - res;
30  }
31  ll que(ll L, ll R, ll k) {
32      return solve(R, k) - solve(L - 1, k);
33  }
```

6.22 幂转下降幂 (求幂和).cpp

```
1   ll comb[N][N];
2   ll s[maxn], inv[maxn], p;
3   // 1^k+2^k+...+n^k
4   void solve() {
5       cin >> k >> n >> p;
6       rep(i, 0, k + 1) {
7           comb[i][0] = comb[i][i] = 1;
8           rep(j, 1, i - 1) {
9               comb[i][j] = (comb[i - 1][j - 1] + comb[i - 1][j]) % p;
10          }
11      }
12      inv[1] = 1;
13      rep(i, 2, k + 1) inv[i] = (p - p / i) * inv[p % i] % p;
14      assert(inv[k] * k % p == 1);
15
16      ll pw = 1;
17      // (k+1)*S[k]=(n+1)^(k+1)-[0-k-1](k+1,j)*S[j]-1
18      rep(i, 0, k) {
19          pw = pw * (n + 1) % p;
20          s[i] = (pw - 1 + p) % p;
21          rep(j, 0, i - 1) {
22              s[i] = (s[i] - comb[i + 1][j] * s[j] % p + p) % p;
23          }
24          s[i] = s[i] * inv[i + 1] % p;
25      }
26      cout << s[k] << '\n';
27  }
```

6.23 扩展欧拉定理.cpp

```
1   // mod {min(b, b % phi + phi)}
2   ll calc(ll p) {
3       if (p == 1) return 0;
4       int phi = p, q = p;
5       for (int i = 2; i * i <= p; i++) {
6           if (q % i == 0) {
7               phi = phi / i * (i - 1);
8               while (q % i == 0) q /= i;
9           }
10      }
11      if (q != 1) phi = phi / q * (q - 1);
12      return powmod(2, calc(phi) + phi, p);
13  }
```

6.24 拉格朗日插值.cpp

```
1   // k阶多项式(需要k+1个点)
2   // 求在点n上的值
3   // O(k)
4   ll lagrange(ll n, int k) {
5       vector<ll> x(k + 5), y(k + 5);
6       rep(i, 1, k + 1) {
7           x[i] = i;
8           // y[i]=(y[i-1]+powmod(i,k-1,mod))%mod;
9       }
10      if (n <= k + 1) return y[n];
11
12      vector<ll> fac(k + 5);
13      fac[0] = 1;
14      ll coe = 1;
15      rep(i, 1, k + 4) fac[i] = fac[i - 1] * i % mod;
16      rep(i, 1, k + 1) coe = coe * (n - i + mod) % mod;
17      ll ans = 0;
18      rep(i, 1, k + 1) {
19          ll sgn = (((k + 1 - i) % 2) ? -1 : 1);
20          ll f1 = powmod(fac[i - 1] * fac[k + 1 - i] % mod, mod - 2, mod);
21          ll f2 = powmod(n - i, mod - 2, mod);
22          ans += sgn * coe * f1 % mod * f2 % mod * y[i] % mod;
23          ans = (ans + mod) % mod;
24      }
25      return ans;
26  }
```

6.25 整除分块.cpp

```cpp
void solve() {
    u64 ans = 0;
    cin >> n;
    for (ll l = 1; l <= n; l++) {
        ll d = n / l, r = n / d;
        ans += (l + r) * (r - l + 1) / 2 * d;
        l = r;
    }
}
```

6.26 枚举子集.cpp

```cpp
void solve() {
    f[0] = 1;
    for (int i = 1; i < (1ll << n); i++) {
        int t = i;
        ll res = 0;
        while (true) {
            if (t == 0) break;
            t = (t - 1)&i;
            res = (res + f[t]) % mod;
        }
        f[i] = res * i;
    }
}
```

6.27 枚举超集.cpp

```cpp
void solve() {
    for (int i = 1; i < (1ll << n); i++) {
        int t = i;
        while (true) {
            t = (t + 1) | i;
            if (t == bit(n) - 1) break;
        }
    }
}
```

6.28 狄利克雷卷积.cpp

```cpp
const int N = 1010000;
```

```cpp
int p[N], pr[N / 5], n, tot;
unsigned int A, B, C, mu[N], f[N], g[N];

inline unsigned int rng61() {
    A ^= A << 16;
    A ^= A >> 5;
    A ^= A << 1;
    unsigned int t = A;
    A = B;
    B = C;
    C ^= t ^ A;
    return C;
}

int main() {
    scanf("%d%u%u%u", &n, &A, &B, &C);
    for (int i = 1; i <= n; i++)
        f[i] = rng61();

    p[1] = 1; mu[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!p[i]) p[i] = i, mu[i] = (uint)-1, pr[++tot] = i;
        for (int j = 1; j <= tot && pr[j] * i <= n; j++) {
            p[i * pr[j]] = pr[j];
            if (p[i] == pr[j]) {
                mu[i * pr[j]] = 0;
                break;
            } else {
                mu[i * pr[j]] = (uint)-mu[i];
            }
        }
    }
    for (int d1 = 1; d1 <= n; d1++)
        for (int d2 = 1; d1 * d2 <= n; d2++)
            g[d1 * d2] += f[d1] * mu[d2];
    uint ans = 0;
    for (int i = 1; i <= n; i++) ans ^= g[i];
    printf("%u\n", ans);
}
```

6.29 线性筛常见积性函数.cpp

```cpp
const int N = 20010000;
```

```
2   int p[N], pr[N / 5], n, pe[N], tot;
3   uint f[N], a, b, ans;
4
5   void compute(int n, function<void(int)> calcpe) {
6       ans = 0;
7       f[1] = 1;
8       for (int i = 2; i <= n; i++) {
9           if (i == pe[i])
10              calcpe(i);
11          else
12              f[i] = f[pe[i]] * f[i / pe[i]];
13      }
14      for (uint i = 1; i <= n; i++) {
15          ans ^= (a * i * f[i] + b);
16      }
17      printf("%u\n", ans);
18  }
19
20  int main() {
21      scanf("%d%u%u", &n, &a, &b);
22      p[1] = 1;
23      for (int i = 2; i <= n; i++) {
24          if (!p[i]) p[i] = i, pe[i] = i, pr[++tot] = i;
25          for (int j = 1; j <= tot && pr[j] * i <= n; j++) {
26              p[i * pr[j]] = pr[j];
27              if (p[i] == pr[j]) {
28                  pe[i * pr[j]] = pe[i] * pr[j];
29                  break;
30              } else {
31                  pe[i * pr[j]] = pr[j];
32              }
33          }
34      }
35      // 因子个数，因子和，欧拉函数，莫比乌斯函数
36      compute(n, [&](int x) {
37          f[x] = f[x / p[x]] + 1;
38      });
39
40      compute(n, [&](int x) {
41          f[x] = f[x / p[x]] + x;
42      });
43
44      compute(n, [&](int x) {
```

```
45          f[x] = x / p[x] * (p[x] - 1);
46      });
47
48      compute(n, [&](int x) {
49          f[x] = x == p[x] ? -1 : 0;
50      });
51  }
```

6.30 莫比乌斯反演 gcd 常见结论.cpp

```
1   // u * 1 = e, phi * 1 = id, phi = id * u
2   const int N = 10100000, M = 10000000;
3   int p[N], pr[N / 5], n, tot;
4   int mu[N], smu[N];
5
6   int main() {
7       p[1] = 1; mu[1] = 1;
8       for (int i = 2; i <= M; i++) {
9           if (!p[i]) p[i] = i, mu[i] = -1, pr[++tot] = i;
10          for (int j = 1; j <= tot && pr[j] * i <= M; j++) {
11              p[i * pr[j]] = pr[j];
12              if (p[i] == pr[j]) {
13                  mu[i * pr[j]] = 0;
14                  break;
15              } else {
16                  mu[i * pr[j]] = -mu[i];
17              }
18          }
19      }
20      for (int i = 1; i <= M; i++)
21          smu[i] = smu[i - 1] + mu[i];
22      int T;
23      scanf("%d", &T);
24      for (int tc = 0; tc < T; tc++) {
25          int n, m;
26          scanf("%d%d", &n, &m);
27          if (n > m) swap(n, m);
28          ll ans = 0;
29          for (int l = 1; l <= n; l++) {
30              int n1 = n / l, m1 = m / l;
31              int r = min(n / n1, m / m1);
32              // l ... r
33              ans += 1ll * (smu[r] - smu[l - 1]) * n1 * m1;
```

```
34              l = r;
35          }
36          printf("%lld\n", ans);
37      }
38  }
```

7 String

7.1 ACAM.cpp

```
1   const int AC_SIGMA = 26, AC_V = 26, AC_N = 810000;
2   struct AC_automaton {
3       struct node {
4           node *go[AC_V], *fail, *f;
5   // declare extra variables:
6       } pool[AC_N], *cur, *root, *q[AC_N];
7       node* newnode() {
8           node *p = cur++;
9   // init extra variables:
10          return p;
11      }
12  // CALL init() and CHECK all const variables:
13      void init() { cur = pool; root = newnode(); }
14      node* append(node *p, int w) {
15          if (!p->go[w]) p->go[w] = newnode(), p->go[w]->f = p;
16          return p->go[w];
17      }
18      void build() {
19          int t = 0;
20          q[t++] = root;
21          root->fail = root;
22          rep(i, 0, AC_SIGMA - 1) if (root->go[i]) {
23              q[t++] = root->go[i];
24              root->go[i]->fail = root;
25          } else {
26              root->go[i] = root;
27          }
28          rep(i, 1, t - 1) {
29              node *u = q[i];
30              rep(j, 0, AC_SIGMA - 1) if (u->go[j]) {
31                  u->go[j]->fail = u->fail->go[j];
32                  q[t++] = u->go[j];
```

```
33              } else {
34                  u->go[j] = u->fail->go[j];
35              }
36          }
37      }
38  } ac;
39  typedef AC_automaton::node ACnode;
40
41  const int M = 2, N = 2.1e5;
42  struct node {
43      node *son[M], *go[M], *fail;
44      int cnt, vis, ins;
45  } pool[N], *cur = pool, *q[N], *root;
46
47  node *newnode() { return cur++; }
48  int t, n;
49
50  void build() {
51      t = 0;
52      q[t++] = root;
53      for (int i = 0; i < t; i++) {
54          node *u = q[i];
55          for (int j = 0; j < M; j++) {
56              if (u->son[j]) {
57                  u->go[j] = u->son[j];
58                  if (u != root)
59                      u->go[j]->fail = u->fail->go[j];
60                  else
61                      u->go[j]->fail = root;
62                  q[t++] = u->son[j];
63              } else {
64                  if (u != root)
65                      u->go[j] = u->fail->go[j];
66                  else
67                      u->go[j] = root;
68              }
69          }
70      }
71  }
72
73  void insert(string &s) {
74      node *cur = root;
75      for (auto c : s) {
```

```cpp
76          int w = c - '0';
77          if (!cur->son[w]) {
78              cur->son[w] = newnode();
79          }
80          cur = cur->son[w];
81      }
82      cur->cnt = 1;
83  }
```

7.2 hash61.cpp

```cpp
1   struct hash61 {
2     static const uint64_t md = (1LL << 61) - 1;
3     static uint64_t step;
4     static vector<uint64_t> pw;
5
6     uint64_t addmod(uint64_t a, uint64_t b) const {
7       a += b;
8       if (a >= md) a -= md;
9       return a;
10    }
11
12    uint64_t submod(uint64_t a, uint64_t b) const {
13      a += md - b;
14      if (a >= md) a -= md;
15      return a;
16    }
17
18    uint64_t mulmod(uint64_t a, uint64_t b) const {
19      uint64_t l1 = (uint32_t) a, h1 = a >> 32, l2 = (uint32_t) b, h2 = b >>
             32;
20      uint64_t l = l1 * l2, m = l1 * h2 + l2 * h1, h = h1 * h2;
21      uint64_t ret = (l & md) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >>
             3) + 1;
22      ret = (ret & md) + (ret >> 61);
23      ret = (ret & md) + (ret >> 61);
24      return ret - 1;
25    }
26
27    void ensure_pw(int sz) {
28      int cur = (int) pw.size();
29      if (cur < sz) {
30        pw.resize(sz);
```

```cpp
31        for (int i = cur; i < sz; i++) {
32          pw[i] = mulmod(pw[i - 1], step);
33        }
34      }
35    }
36
37    vector<uint64_t> pref;
38    int n;
39
40    template<typename T>
41    hash61(const T& s) {
42      n = (int) s.size();
43      ensure_pw(n + 1);
44      pref.resize(n + 1);
45      pref[0] = 1;
46      for (int i = 0; i < n; i++) {
47        pref[i + 1] = addmod(mulmod(pref[i], step), s[i]);
48      }
49    }
50
51    inline uint64_t operator()(const int from, const int to) const {
52      assert(0 <= from && from <= to && to <= n - 1);
53      return submod(pref[to + 1], mulmod(pref[from], pw[to - from + 1]));
54    }
55  };
56
57  uint64_t hash61::step = (md >> 2) + rng() % (md >> 1);
58  vector<uint64_t> hash61::pw = vector<uint64_t>(1, 1);
```

7.3 kmp.cpp

```cpp
1   template <typename T>
2   vector<int> kmp_table(int n, const T &s) {
3     vector<int> p(n, 0);
4     int k = 0;
5     for (int i = 1; i < n; i++) {
6       while (k > 0 && !(s[i] == s[k])) {
7         k = p[k - 1];
8       }
9       if (s[i] == s[k]) {
10        k++;
11      }
12      p[i] = k;
```

```
13    }
14    return p;
15  }
16
17  template <typename T>
18  vector<int> kmp_table(const T &s) {
19    return kmp_table((int) s.size(), s);
20  }
21
22  template <typename T>
23  vector<int> kmp_search(int n, const T &s, int m, const T &w, const vector<
        int> &p) {
24    assert(n >= 1 && (int) p.size() == n);
25    vector<int> res;
26    int k = 0;
27    for (int i = 0; i < m; i++) {
28      while (k > 0 && (k == n || !(w[i] == s[k]))) {
29        k = p[k - 1];
30      }
31      if (w[i] == s[k]) {
32        k++;
33      }
34      if (k == n) {
35        res.push_back(i - n + 1);
36      }
37    }
38    return res;
39    // returns 0-indexed positions of occurrences of s in w
40  }
41
42  template <typename T>
43  vector<int> kmp_search(const T &s, const T &w, const vector<int> &p) {
44    return kmp_search((int) s.size(), s, (int) w.size(), w, p);
45  }
```

7.4   manacherfast.cpp

```
1  template <typename T>
2  vector<int> manacher(int n, const T &s) {
3    if (n == 0) {
4      return vector<int>();
5    }
6    vector<int> res(2 * n - 1, 0);
```

```
7    int l = -1, r = -1;
8    for (int z = 0; z < 2 * n - 1; z++) {
9      int i = (z + 1) >> 1;
10      int j = z >> 1;
11      int p = (i >= r ? 0 : min(r - i, res[2 * (l + r) - z]));
12      while (j + p + 1 < n && i - p - 1 >= 0) {
13        if (!(s[j + p + 1] == s[i - p - 1])) {
14          break;
15        }
16        p++;
17      }
18      if (j + p > r) {
19        l = i - p;
20        r = j + p;
21      }
22      res[z] = p;
23    }
24    return res;
25    // res[2 * i] = odd radius in position i
26    // res[2 * i + 1] = even radius between positions i and i + 1
27    // s = "abaa" -> res = {0, 0, 1, 0, 0, 1, 0}
28    // in other words, for every z from 0 to 2 * n - 2:
29    // calculate i = (z + 1) >> 1 and j = z >> 1
30    // now there is a palindrome from i - res[z] to j + res[z]
31    // (watch out for i > j and res[z] = 0)
32  }
33
34  template <typename T>
35  vector<int> manacher(const T &s) {
36    return manacher((int) s.size(), s);
37  }
```

7.5   MinRotation.cpp

```
1  Description: Finds the lexicographically smallest rotation of a string.
2  Usage: rotate(v.begin(), v.begin() + minRotation(v), v.end());
3  Time: O (N)
4
5  int minRotation(string s) {
6      int a = 0, N = sz(s); s += s;
7      rep(b, 0, N) rep(k, 0, N) {
8          if (a + k == b || s[a + k] < s[b + k]) {b += max(0, k - 1); break;}
9          if (s[a + k] > s[b + k]) { a = b; break; }
```

```
10        }
11        return a;
12   }
```

7.6 PAM.cpp

```
1  struct PAM {
2      struct T {
3          array<int, 10> tr;
4          int fail, len, tag;
5          T() : fail(0), len(0), tag(0) {
6              tr.fill(0);
7          }
8      };
9      vector<T> t;
10     vector<int> stk;
11     int newnode(int len) {
12         t.emplace_back();
13         t.back().len = len;
14         return (int)t.size() - 1;
15     }
16     PAM() : t(2) {
17         t[0].fail = 1, t[0].len = 0;
18         t[1].fail = 0, t[1].len = -1;
19         stk.push_back(-1);
20     }
21     int getfail(int v) {
22         while (stk.end()[-2 - t[v].len] != stk.back()) {
23             v = t[v].fail;
24         }
25         return v;
26     }
27     int insert(int lst, int c, int td) {
28         stk.emplace_back(c);
29         int x = getfail(lst);
30         if (!t[x].tr[c]) {
31             int u = newnode(t[x].len + 2);
32             t[u].fail = t[getfail(t[x].fail)].tr[c];
33             t[x].tr[c] = u;
34         }
35         t[t[x].tr[c]].tag += td;
36         return t[x].tr[c];
37     }
```

```
38     int build(int n) {
39         int ans = 0;
40         for (int i = (int)t.size() - 1; i > 1; i--) {
41             t[t[i].fail].tag += t[i].tag;
42             if (t[i].len > n) {
43                 continue;
44             }
45             ans = (ans + 1ll * t[i].tag * t[i].tag % M * t[i].len) % M;
46         }
47         return ans;
48     }
49 };
```

7.7 rollingHash.cpp

```
1  typedef pair<int,int> hashv;
2  const ll mod1=1000000007;
3  const ll mod2=1000000009;
4
5  // prefixSum trick for high dimensions
6
7  hashv operator + (hashv a,hashv b) {
8      int c1=a.fi+b.fi,c2=a.se+b.se;
9      if (c1>=mod1) c1-=mod1;
10     if (c2>=mod2) c2-=mod2;
11     return mp(c1,c2);
12 }
13
14 hashv operator - (hashv a,hashv b) {
15     int c1=a.fi-b.fi,c2=a.se-b.se;
16     if (c1<0) c1+=mod1;
17     if (c2<0) c2+=mod2;
18     return mp(c1,c2);
19 }
20
21 hashv operator * (hashv a,hashv b) {
22     return mp(1ll*a.fi*b.fi%mod1,1ll*a.se*b.se%mod2);
23 }
```

7.8 SA.cpp

```
1  #include <bits/stdc++.h>
2  using namespace std;
```

```cpp
 3
 4   const int N = 101000;
 5   char s[N];
 6   int sa[N], rk[N], ht[N], n;
 7   // 0-based sa 表示第 i 大的为哪个, rk 表示第 i 个后缀第几大
 8   // ht表示 lcp(sa[i], sa[i-1])
 9   void buildSA(char *s, int *sa, int *rk, int *ht, int n, int m = 128) {
10       static int x[N], y[N], c[N];
11       s[n] = 0;
12       for (int i = 0; i < m; i++) c[i] = 0;
13       for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
14       for (int i = 1; i < m; i++) c[i] += c[i - 1];
15       for (int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
16       for (int k = 1; k < n; k <<= 1) {
17           int p=0;
18           for (int i = n - 1; i >= n - k; i--) y[p++] = i;
19           for (int i = 0; i < n; i++) if (sa[i] >= k) y[p++] = sa[i] - k;
20           for (int i = 0; i < m; i++) c[i] = 0;
21           for (int i = 0; i < n; i++) c[x[y[i]]]++;
22           for (int i = 1; i < m; i++) c[i] += c[i - 1];
23           for (int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
24           swap(x, y);
25           p = 1; x[sa[0]] = 0; y[n] = -1;
26           for (int i = 1; i < n; i++) {
27               if (y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k
                     ])
28                   x[sa[i]] = p - 1;
29               else
30                   x[sa[i]] = p++;
31           }
32           if (p == n) break;
33           m = p;
34       }
35       for (int i = 0; i < n; i++) rk[sa[i]] = i;
36       int k = 0;
37       for (int i = 0; i < n; i++) {
38           k = max(k - 1, 0);
39           if (rk[i] == 0) continue;
40           int j = sa[rk[i] - 1];
41           while (s[i + k] == s[j + k]) k++;
42           ht[rk[i]] = k;
43       }
44   }
```

```cpp
45
46   int LCP(int u, int v) {
47       if (u == v) return n - u;
48       if (rk[u] > rk[v]) swap(u, v);
49       // RMQ(ht, rk[u] + 1, rk[v])
50   }
51
52   int main() {
53       scanf("%s", s);
54       n = strlen(s);
55       buildSA(s, sa, rk, ht, n);
56       for (int i = 0; i < n; i++) printf("%d␣", sa[i] + 1); puts("");
57       for (int i = 1; i < n; i++) printf("%d␣", ht[i]); puts("");
58   }
```

7.9   SAfast.cpp

```cpp
 1   template <typename T>
 2   vector<int> suffix_array(int n, const T &s, int char_bound) {
 3     vector<int> a(n);
 4     if (n == 0) {
 5       return a;
 6     }
 7     if (char_bound != -1) {
 8       vector<int> aux(char_bound, 0);
 9       for (int i = 0; i < n; i++) {
10         aux[s[i]]++;
11       }
12       int sum = 0;
13       for (int i = 0; i < char_bound; i++) {
14         int add = aux[i];
15         aux[i] = sum;
16         sum += add;
17       }
18       for (int i = 0; i < n; i++) {
19         a[aux[s[i]]++] = i;
20       }
21     } else {
22       iota(a.begin(), a.end(), 0);
23       sort(a.begin(), a.end(), [&s](int i, int j) { return s[i] < s[j]; });
24     }
25     vector<int> sorted_by_second(n);
26     vector<int> ptr_group(n);
```

```
27      vector<int> new_group(n);
28      vector<int> group(n);
29      group[a[0]] = 0;
30      for (int i = 1; i < n; i++) {
31        group[a[i]] = group[a[i - 1]] + (!(s[a[i]] == s[a[i - 1]]));
32      }
33      int cnt = group[a[n - 1]] + 1;
34      int step = 1;
35      while (cnt < n) {
36        int at = 0;
37        for (int i = n - step; i < n; i++) {
38          sorted_by_second[at++] = i;
39        }
40        for (int i = 0; i < n; i++) {
41          if (a[i] - step >= 0) {
42            sorted_by_second[at++] = a[i] - step;
43          }
44        }
45        for (int i = n - 1; i >= 0; i--) {
46          ptr_group[group[a[i]]] = i;
47        }
48        for (int i = 0; i < n; i++) {
49          int x = sorted_by_second[i];
50          a[ptr_group[group[x]]++] = x;
51        }
52        new_group[a[0]] = 0;
53        for (int i = 1; i < n; i++) {
54          if (group[a[i]] != group[a[i - 1]]) {
55            new_group[a[i]] = new_group[a[i - 1]] + 1;
56          } else {
57            int pre = (a[i - 1] + step >= n ? -1 : group[a[i - 1] + step]);
58            int cur = (a[i] + step >= n ? -1 : group[a[i] + step]);
59            new_group[a[i]] = new_group[a[i - 1]] + (pre != cur);
60          }
61        }
62        swap(group, new_group);
63        cnt = group[a[n - 1]] + 1;
64        step <<= 1;
65      }
66      return a;
67  }
68
69  template <typename T>
70  vector<int> suffix_array(const T &s, int char_bound) {
71      return suffix_array((int) s.size(), s, char_bound);
72  }
73
74  template <typename T>
75  vector<int> build_lcp(int n, const T &s, const vector<int> &sa) {
76      assert((int) sa.size() == n);
77      vector<int> pos(n);
78      for (int i = 0; i < n; i++) {
79        pos[sa[i]] = i;
80      }
81      vector<int> lcp(max(n - 1, 0));
82      int k = 0;
83      for (int i = 0; i < n; i++) {
84        k = max(k - 1, 0);
85        if (pos[i] == n - 1) {
86          k = 0;
87        } else {
88          int j = sa[pos[i] + 1];
89          while (i + k < n && j + k < n && s[i + k] == s[j + k]) {
90            k++;
91          }
92          lcp[pos[i]] = k;
93        }
94      }
95      return lcp;
96  }
97
98  template <typename T>
99  vector<int> build_lcp(const T &s, const vector<int> &sa) {
100     return build_lcp((int) s.size(), s, sa);
101 }
```

7.10   SAM.cpp

```
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       };
```

```cpp
 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    };
```

7.11 SA-IS.cpp

```cpp
 1  /*
 2   * Time Complexity: Suffix Array: O(N + Character_Set_Size) time and space
 *    //
 3   128 --- ASCII
 4   *                    LCP: O(N) time and space
 5   * Usage:
 6   *        1. Suffix Array (returns s.size() elements, NOT considering
 7   0-length/empty suffix)
 8   *              auto sa = suffix_array(s); // s is the input string with
 *    ASCII
 9   characters
10   *              auto sa_wide_char = suffix_array(s, LIM); // LIM = max(s[i])
 *    + 2,
11   s is the string with arbitary big characters.
12   *        2. LCP:
13   *              auto lcp = LCP(s, suffix_array(s)); // returns s.size()
 *    elements,
14   where lcp[i]=LCP(sa[i], sa[i+1])
15   * Status: Tested (DMOJ: ccc03s4, SPOJ: SARRAY (100pts), Yosupo's: Suffix
 *    Array
16   & Number of Substrings, CodeForces EDU
17   */
18  // Based on: Rickypon, https://judge.yosupo.jp/submission/10105
19  void induced_sort(const std::vector<int>& vec, int val_range,
20                    std::vector<int>& SA, const std::vector<bool>& sl,
21                    const std::vector<int>& lms_idx) {
22      std::vector<int> l(val_range, 0), r(val_range, 0);
23      for (int c : vec) {
24          if (c + 1 < val_range) ++l[c + 1];
25          ++r[c];
26      }
27      std::partial_sum(l.begin(), l.end(), l.begin());
28      std::partial_sum(r.begin(), r.end(), r.begin());
29      std::fill(SA.begin(), SA.end(), -1);
30      for (int i = (int)lms_idx.size() - 1; i >= 0; --i)
31          SA[--r[vec[lms_idx[i]]]] = lms_idx[i];
32      for (int i : SA)
33          if (i >= 1 && sl[i - 1]) SA[l[vec[i - 1]]++] = i - 1;
34      std::fill(r.begin(), r.end(), 0);
35      for (int c : vec) ++r[c];
36      std::partial_sum(r.begin(), r.end(), r.begin());
37      for (int k = (int)SA.size() - 1, i = SA[k]; k >= 1; --k, i = SA[k])
38          if (i >= 1 && !sl[i - 1]) {
```

```
39            SA[--r[vec[i - 1]]] = i - 1;
40        }
41 }
42
43 std::vector<int> SA_IS(const std::vector<int>& vec, int val_range) {
44     const int n = vec.size();
45     std::vector<int> SA(n), lms_idx;
46     std::vector<bool> sl(n);
47     sl[n - 1] = false;
48     for (int i = n - 2; i >= 0; --i) {
49         sl[i] = (vec[i] > vec[i + 1] || (vec[i] == vec[i + 1] && sl[i + 1]))
                   ;
50         if (sl[i] && !sl[i + 1]) lms_idx.push_back(i + 1);
51     }
52     std::reverse(lms_idx.begin(), lms_idx.end());
53     induced_sort(vec, val_range, SA, sl, lms_idx);
54     std::vector<int> new_lms_idx(lms_idx.size()), lms_vec(lms_idx.size());
55     for (int i = 0, k = 0; i < n; ++i)
56         if (!sl[SA[i]] && SA[i] >= 1 && sl[SA[i] - 1]) {
57             new_lms_idx[k++] = SA[i];
58         }
59     int cur = 0;
60     SA[n - 1] = cur;
61     for (size_t k = 1; k < new_lms_idx.size(); ++k) {
62         int i = new_lms_idx[k - 1], j = new_lms_idx[k];
63         if (vec[i] != vec[j]) {
64             SA[j] = ++cur;
65             continue;
66         }
67         bool flag = false;
68         for (int a = i + 1, b = j + 1;; ++a, ++b) {
69             if (vec[a] != vec[b]) {
70                 flag = true;
71                 break;
72             }
73             if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1])) {
74                 flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b - 1]));
75                 break;
76             }
77         }
78         SA[j] = (flag ? ++cur : cur);
79     }
80     for (size_t i = 0; i < lms_idx.size(); ++i) lms_vec[i] = SA[lms_idx[i]];
```

```
81     if (cur + 1 < (int)lms_idx.size()) {
82         auto lms_SA = SA_IS(lms_vec, cur + 1);
83         for (size_t i = 0; i < lms_idx.size(); ++i) {
84             new_lms_idx[i] = lms_idx[lms_SA[i]];
85         }
86     }
87     induced_sort(vec, val_range, SA, sl, new_lms_idx);
88     return SA;
89 }
90
91 std::vector<int> suffix_array(const std::string& s, const char first = 'a',
92                               const char last = 'z') {
93     std::vector<int> vec(s.size() + 1);
94     std::copy(std::begin(s), std::end(s), std::begin(vec));
95     for (auto& x : vec) x -= (int)first - 1;
96     vec.back() = 0;
97     auto ret = SA_IS(vec, (int)last - (int)first + 2);
98     ret.erase(ret.begin());
99     return ret;
100 }
101 // Author: https://codeforces.com/blog/entry/12796?#comment-175287
102 // Uses kasai's algorithm linear in time and space
103 std::vector<int> LCP(const std::string& s, const std::vector<int>& sa) {
104     int n = s.size(), k = 0;
105     std::vector<int> lcp(n), rank(n);
106     for (int i = 0; i < n; i++) rank[sa[i]] = i;
107     for (int i = 0; i < n; i++, k ? k-- : 0) {
108         if (rank[i] == n - 1) {
109             k = 0;
110             continue;
111         }
112         int j = sa[rank[i] + 1];
113         while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
114         lcp[rank[i]] = k;
115     }
116     lcp[n - 1] = 0;
117     return lcp;
118 }
119
120 template <typename T, class F = function<T(const T&, const T&)>>
121 class SparseTable {
122  public:
123     int n;
```

```
124    vector<vector<T>> mat;
125    F func;
126
127    SparseTable(const vector<T>& a, const F& f) : func(f) {
128      n = static_cast<int>(a.size());
129      int max_log = 32 - __builtin_clz(n);
130      mat.resize(max_log);
131      mat[0] = a;
132      for (int j = 1; j < max_log; j++) {
133        mat[j].resize(n - (1 << j) + 1);
134        for (int i = 0; i <= n - (1 << j); i++) {
135          mat[j][i] = func(mat[j - 1][i], mat[j - 1][i + (1 << (j - 1))]);
136        }
137      }
138    }
139
140    T get(int from, int to) const {
141      assert(0 <= from && from <= to && to <= n - 1);
142      int lg = 32 - __builtin_clz(to - from + 1) - 1;
143      return func(mat[lg][from], mat[lg][to - (1 << lg) + 1]);
144    }
145  };
```

7.12   Z.cpp

```
1   template <typename T>
2   vector<int> z_function(int n, const T &s) {
3     vector<int> z(n, n);
4     int l = 0, r = 0;
5     for (int i = 1; i < n; i++) {
6       z[i] = (i > r ? 0 : min(r - i + 1, z[i - l]));
7       while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
8         z[i]++;
9       }
10      if (i + z[i] - 1 > r) {
11        l = i;
12        r = i + z[i] - 1;
13      }
14    }
15    return z;
16  }
17
18  template <typename T>
```

```
19  vector<int> z_function(const T &s) {
20    return z_function((int) s.size(), s);
21  }
```

8   Basic

8.1   AST.py

```
1   class Solution:
2       def calculate(self, s: str) -> int:
3           sign = ['+', '-', '*', '/', '(', ')']
4           v = []
5           num = ''
6           for c in s:
7               if c in sign:
8                   if num:
9                       v.append(num)
10                      num = ''
11                  if c == '-' and (not v or v[-1] == '('):
12                      v.append('0')
13                  v.append(c)
14              elif c.isnumeric():
15                  num += c
16          if num:
17              v.append(num)
18
19          stk0 = []
20          stk1 = []
21          for e in v:
22              if e.isnumeric():
23                  stk0.append(e)
24              elif e in ['+', '-']:
25                  while stk1 and stk1[-1] in ['*', '/', '+', '-']:
26                      stk0.append(stk1.pop())
27                  stk1.append(e)
28              elif e in ['*', '/', '(']:
29                  stk1.append(e)
30              else:
31                  while stk1 and stk1[-1] != '(':
32                      stk0.append(stk1.pop())
33                  stk1.pop()
34          while stk1:
```

```
35              stk0.append(stk1.pop())
36
37          res = []
38          for e in stk0:
39              if e.isnumeric():
40                  res.append(int(e))
41              else:
42                  v = res.pop()
43                  u = res.pop()
44                  if e == '+':
45                      res.append(u + v)
46                  if e == '-':
47                      res.append(u - v)
48                  if e == '*':
49                      res.append(u * v)
50                  if e == '/':
51                      res.append(u // v)
52          return res[0]
```

8.2   bitset.cpp

```
1   template <int len = 1>
2   void solve(int n) {
3       if (n > len) {
4           solve<std::min(len*2, MAXLEN)>(n);
5           return;
6       }
7       // solution using bitset<len>
8   }
9
10  struct Bitset {
11      vector<ull> b;
12      int n;
13      Bitset(int x = 0) {
14          n = x;
15          b.resize((n + 63) / 64, 0);
16      }
17
18      int get(int x) {
19          return (b[x >> 6] >> (x & 63)) & 1;
20      }
21
22      void set(int x, int y) {
```

```
23          b[x >> 6] |= 1ULL << (x & 63);
24          if (!y) b[x >> 6] ^= 1ULL << (x & 63);
25      }
26
27      Bitset &operator&=(const Bitset &another) {
28          rep(i, 0, min(SZ(b), SZ(another.b)) - 1) {
29              b[i] &= another.b[i];
30          }
31          return (*this);
32      }
33
34      Bitset operator&(const Bitset &another)const {
35          return (Bitset(*this) &= another);
36      }
37
38      Bitset &operator|=(const Bitset &another) {
39          rep(i, 0, min(SZ(b), SZ(another.b)) - 1) {
40              b[i] |= another.b[i];
41          }
42          return (*this);
43      }
44
45      Bitset operator|(const Bitset &another)const {
46          return (Bitset(*this) |= another);
47      }
48
49      Bitset &operator^=(const Bitset &another) {
50          rep(i, 0, min(SZ(b), SZ(another.b)) - 1) {
51              b[i] ^= another.b[i];
52          }
53          return (*this);
54      }
55
56      Bitset operator^(const Bitset &another)const {
57          return (Bitset(*this) ^= another);
58      }
59
60      Bitset &operator>>=(int x) {
61          if (x & 63) {
62              rep(i, 0, SZ(b) - 2) {
63                  b[i] >>= (x & 63);
64                  b[i] ^= (b[i + 1] << (64 - (x & 63)));
65              }
```

```
66                b.back() >>= (x & 63);
67            }
68
69        x >>= 6;
70        rep(i, 0, SZ(b) - 1) {
71            if (i + x < SZ(b)) b[i] = b[i + x];
72            else b[i] = 0;
73        }
74        return (*this);
75    }
76
77    Bitset operator>>(int x)const {
78        return (Bitset(*this) >>= x);
79    }
80
81    Bitset &operator<<=(int x) {
82        if (x & 63) {
83            for (int i = SZ(b) - 1; i >= 1; i--) {
84                b[i] <<= (x & 63);
85                b[i] ^= b[i - 1] >> (64 - (x & 63));
86            }
87            b[0] <<= x & 63;
88        }
89
90        x >>= 6;
91        for (int i = SZ(b) - 1; i >= 0; i--) {
92            if (i - x >= 0) b[i] = b[i - x];
93            else b[i] = 0;
94        }
95        return (*this);
96    }
97
98    Bitset operator<<(int x)const {
99        return (Bitset(*this) <<= x);
100    }
101 };
```

8.3   fastIO.cpp

```
1  static struct FastInput {
2    static constexpr int BUF_SIZE = 1 << 20;
3    char buf[BUF_SIZE];
4    size_t chars_read = 0;
5    size_t buf_pos = 0;
6    FILE *in = stdin;
7    char cur = 0;
8
9    inline char get_char() {
10     if (buf_pos >= chars_read) {
11       chars_read = fread(buf, 1, BUF_SIZE, in);
12       buf_pos = 0;
13       buf[0] = (chars_read == 0 ? -1 : buf[0]);
14     }
15     return cur = buf[buf_pos++];
16   }
17
18   template <typename T>
19   inline void tie(T) {}
20
21   inline explicit operator bool() {
22     return cur != -1;
23   }
24
25   inline static bool is_blank(char c) {
26     return c <= ' ';
27   }
28
29   inline bool skip_blanks() {
30     while (is_blank(cur) && cur != -1) {
31       get_char();
32     }
33     return cur != -1;
34   }
35
36   inline FastInput& operator>>(char& c) {
37     skip_blanks();
38     c = cur;
39     get_char();
40     return *this;
41   }
42
43   inline FastInput& operator>>(string& s) {
44     if (skip_blanks()) {
45       s.clear();
46       do {
47         s += cur;
```

```
48        } while (!is_blank(get_char()));
49      }
50      return *this;
51    }
52
53    template <typename T>
54    inline FastInput& read_integer(T& n) {
55      // unsafe, doesn't check that characters are actually digits
56      n = 0;
57      if (skip_blanks()) {
58        int sign = +1;
59        if (cur == '-') {
60          sign = -1;
61          get_char();
62        }
63        do {
64          n += n + (n << 3) + cur - '0';
65        } while (!is_blank(get_char()));
66        n *= sign;
67      }
68      return *this;
69    }
70
71    template <typename T>
72    inline typename enable_if<is_integral<T>::value, FastInput&>::type
73        operator>>(T& n) {
74      return read_integer(n);
75    }
76
77    #if !defined(_WIN32) || defined(_WIN64)
78    inline FastInput& operator>>(__int128& n) {
79      return read_integer(n);
80    }
81    #endif
82
83    template <typename T>
84    inline typename enable_if<is_floating_point<T>::value, FastInput&>::type
85        operator>>(T& n) {
86      // not sure if really fast, for compatibility only
87      n = 0;
88      if (skip_blanks()) {
89        string s;
90        (*this) >> s;
```

```
89        sscanf(s.c_str(), "%lf", &n);
90      }
91      return *this;
92    }
93  } fast_input;
94
95  #define cin fast_input
96
97  static struct FastOutput {
98    static constexpr int BUF_SIZE = 1 << 20;
99    char buf[BUF_SIZE];
100   size_t buf_pos = 0;
101   static constexpr int TMP_SIZE = 1 << 20;
102   char tmp[TMP_SIZE];
103   FILE *out = stdout;
104
105   inline void put_char(char c) {
106     buf[buf_pos++] = c;
107     if (buf_pos == BUF_SIZE) {
108       fwrite(buf, 1, buf_pos, out);
109       buf_pos = 0;
110     }
111   }
112
113   ~FastOutput() {
114     fwrite(buf, 1, buf_pos, out);
115   }
116
117   inline FastOutput& operator<<(char c) {
118     put_char(c);
119     return *this;
120   }
121
122   inline FastOutput& operator<<(const char* s) {
123     while (*s) {
124       put_char(*s++);
125     }
126     return *this;
127   }
128
129   inline FastOutput& operator<<(const string& s) {
130     for (int i = 0; i < (int) s.size(); i++) {
131       put_char(s[i]);
```

```
132        }
133      return *this;
134    }
135
136    template <typename T>
137    inline char* integer_to_string(T n) {
138      // beware of TMP_SIZE
139      char* p = tmp + TMP_SIZE - 1;
140      if (n == 0) {
141        *--p = '0';
142      } else {
143        bool is_negative = false;
144        if (n < 0) {
145          is_negative = true;
146          n = -n;
147        }
148        while (n > 0) {
149          *--p = (char) ('0' + n % 10);
150          n /= 10;
151        }
152        if (is_negative) {
153          *--p = '-';
154        }
155      }
156      return p;
157    }
158
159    template <typename T>
160    inline typename enable_if<is_integral<T>::value, char*>::type stringify(T
           n) {
161      return integer_to_string(n);
162    }
163
164    #if !defined(_WIN32) || defined(_WIN64)
165    inline char* stringify(__int128 n) {
166      return integer_to_string(n);
167    }
168    #endif
169
170    template <typename T>
171    inline typename enable_if<is_floating_point<T>::value, char*>::type
           stringify(T n) {
172      sprintf(tmp, "%.17f", n);
```

```
173      return tmp;
174    }
175
176    template <typename T>
177    inline FastOutput& operator<<(const T& n) {
178      auto p = stringify(n);
179      for (; *p != 0; p++) {
180        put_char(*p);
181      }
182      return *this;
183    }
184  } fast_output;
185
186  #define cout fast_output
```

8.4 FastMod.cpp

```
1   Description: Compute a % b about 5 times faster than usual, where b is
2   constant but not known at compile time. Returns a value congruent to a
3   (mod b) in the range [0, 2b).
4
5   typedef unsigned long long ull;
6   struct FastMod {
7       ull b, m;
8       FastMod(ull b) : b(b), m(-1ULL / b) {}
9       ull reduce(ull a) { // a % b + (0 or b)
10          return a - (ull)((__uint128_t(m) * a) >> 64) * b;
11      }
12  };
```

8.5 intervalContainer.cpp

```
1   Description: Add and remove intervals from a set of disjoint intervals.
2   Will merge the added interval with any overlapping intervals in the set when
3   adding. Intervals are [inclusive, exclusive).
4   Time: O (log N)
5
6   set<pii>::iterator addInterval(set<pii>& is, int L, int R) {
7       if (L == R) return is.end();
8       auto it = is.lower_bound({L, R}), before = it;
9       while (it != is.end() && it->first <= R) {
10          R = max(R, it->second);
11          before = it = is.erase(it);
```

```
12            }
13        if (it != is.begin() && (--it)->second >= L) {
14            L = min(L, it->first);
15            R = max(R, it->second);
16            is.erase(it);
17        }
18        return is.insert(before, {L, R});
19    }
20    void removeInterval(set<pii>& is, int L, int R) {
21        if (L == R) return;
22        auto it = addInterval(is, L, R);
23        auto r2 = it->second;
24        if (it->first == L) is.erase(it);
25        else (int&)it->second = L;
26        if (R != r2) is.emplace(R, r2);
27    }
```

8.6   lineContainer.cpp

```
1    /**
2     * Author: Simon Lindholm
3     * Date: 2017-04-20
4     * License: CC0
5     * Source: own work
6     * Description: Container where you can add lines of the form kx+m, and
7          query maximum values at points x.
8     *  Useful for dynamic programming (``convex hull trick'').
9     * Time: O(\log N)
10    * Status: stress-tested
11    */
12    #pragma once
13
14    struct Line {
15        mutable ll k, m, p;
16        bool operator<(const Line& o) const { return k < o.k; }
17        bool operator<(ll x) const { return p < x; }
18    };
19
20    struct LineContainer : multiset<Line, less<>> {
21        // (for doubles, use inf = 1/.0, div(a,b) = a/b)
22        static const ll inf = LLONG_MAX;
23        ll div(ll a, ll b) { // floored division
24            return a / b - ((a ^ b) < 0 && a % b); }
```

```
24        bool isect(iterator x, iterator y) {
25            if (y == end()) return x->p = inf, 0;
26            if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
27            else x->p = div(y->m - x->m, x->k - y->k);
28            return x->p >= y->p;
29        }
30        void add(ll k, ll m) {
31            auto z = insert({k, m, 0}), y = z++, x = y;
32            while (isect(y, z)) z = erase(z);
33            if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
34            while ((y = x) != begin() && (--x)->p >= y->p)
35                isect(x, erase(y));
36        }
37        ll query(ll x) {
38            assert(!empty());
39            auto l = *lower_bound(x);
40            return l.k * x + l.m;
41        }
42    };
```

8.7   mint.cpp

```
1    template<int MOD, int RT> struct mint {
2        static const int mod = MOD;
3        static constexpr mint rt() { return RT; } // primitive root for FFT
4        int v; explicit operator int() const { return v; } // explicit -> don't
5            silently convert to int
5        mint():v(0) {}
6        mint(ll _v) { v = int((-MOD < _v && _v < MOD) ? _v : _v % MOD);
7            if (v < 0) v += MOD; }
8        bool operator==(const mint& o) const {
9            return v == o.v; }
10        friend bool operator!=(const mint& a, const mint& b) {
11            return !(a == b); }
12        friend bool operator<(const mint& a, const mint& b) {
13            return a.v < b.v; }
14
15        mint& operator+=(const mint& o) {
16            if ((v += o.v) >= MOD) v -= MOD;
17            return *this; }
18        mint& operator-=(const mint& o) {
19            if ((v -= o.v) < 0) v += MOD;
20            return *this; }
```

```cpp
21      mint& operator*=(const mint& o) {
22          v = int((ll)v*o.v%MOD); return *this; }
23      mint& operator/=(const mint& o) { return (*this) *= inv(o); }
24      friend mint pow(mint a, ll p) {
25          mint ans = 1; assert(p >= 0);
26          for (; p; p /= 2, a *= a) if (p&1) ans *= a;
27          return ans; }
28      friend mint inv(const mint& a) { assert(a.v != 0);
29          return pow(a,MOD-2); }
30
31      mint operator-() const { return mint(-v); }
32      mint& operator++() { return *this += 1; }
33      mint& operator--() { return *this -= 1; }
34      friend mint operator+(mint a, const mint& b) { return a += b; }
35      friend mint operator-(mint a, const mint& b) { return a -= b; }
36      friend mint operator*(mint a, const mint& b) { return a *= b; }
37      friend mint operator/(mint a, const mint& b) { return a /= b; }
38  };
39
40  const int MOD=998244353;
41  using mi = mint<MOD,5>; // 5 is primitive root for both common mods
42
43  namespace simp {
44      vector<mi> fac,ifac,invn;
45      void check(int x) {
46          if (fac.empty()) {
47              fac={mi(1),mi(1)};
48              ifac={mi(1),mi(1)};
49              invn={mi(0),mi(1)};
50          }
51          while (SZ(fac)<=x) {
52              int n=SZ(fac),m=SZ(fac)*2;
53              fac.resize(m);
54              ifac.resize(m);
55              invn.resize(m);
56              for (int i=n;i<m;i++) {
57                  fac[i]=fac[i-1]*mi(i);
58                  invn[i]=mi(MOD-MOD/i)*invn[MOD%i];
59                  ifac[i]=ifac[i-1]*invn[i];
60              }
61          }
62      }
63      mi gfac(int x) {
64          check(x); return fac[x];
65      }
66      mi ginv(int x) {
67          check(x); return invn[x];
68      }
69      mi gifac(int x) {
70          check(x); return ifac[x];
71      }
72      mi binom(int n,int m) {
73          if (m < 0 || m > n) return mi(0);
74          return gfac(n)*gifac(m)*gifac(n - m);
75      }
76  }
```

8.8   pbds.cpp

```cpp
1   #include <bits/extc++.h>
2   using namespace __gnu_cxx;
3   using namespace __gnu_pbds;
4
5   #include<ext/pb_ds/assoc_container.hpp>
6   #include<ext/pb_ds/tree_policy.hpp>
7   #include<ext/pb_ds/hash_policy.hpp>
8   #include<ext/pb_ds/trie_policy.hpp>
9   #include<ext/pb_ds/priority_queue.hpp>
10
11  pairing_heap_tag: 配对堆
12  thin_heap_tag: 斐波那契堆
13  binomial_heap_tag: 二项堆
14  binary_heap_tag: 二叉堆
15
16  __gnu_pbds::priority_queue<PII, greater<PII>, pairing_heap_tag> q;
17  __gnu_pbds::priority_queue<PII, greater<PII>, pairing_heap_tag>::
        point_iterator its[N];
18
19  its[v] = q.push({dis[v], v});
20  q.modify(its[v], {dis[v], v});
21
22  可以将两个优先队列中的元素合并（无任何约束）
23  使用方法为a.join(b)
24  此时优先队列b内所有元素就被合并进优先队列a中，且优先队列b被清空
25
26  cc_hash_table<string, int> mp1拉链法
```

```
27   gp_hash_table<string, int> mp2查探法
```

8.9   simu.cpp

```
1   db rnd(db l, db r) {
2     static uniform_real_distribution<db> u(0, 1);
3     static default_random_engine e(rng());
4     return l + (r - l) * u(e);   // u(rng);
5   }
6
7   db eval(pair<db, db> x) { ... }
8
9   void simulate_anneal() {
10    pair<db, db> cur(rnd(0, 10000), rnd(0, 10000));
11    for (double k = 10000; k > 1e-5; k *= 0.99) {
12      // [start, end, step]
13      pair<db, db> nxt(cur.fi + rnd(-k, k), cur.se + rnd(-k, k));
14      db delta = eval(nxt) - eval(cur);
15      if (exp(-delta / k) > rnd(0, 1)) {
16        cur = nxt;
17      }
18    }
19  }
20
21  /**
22   * https://codeforces.com/gym/104813/submission/234982955
23   * The 9th CCPC (Harbin) 2023
24   * Author: QwertyPi
25   */
26  LD Prob() {
27    static uniform_real_distribution<> dist(0.0, 1.0);
28    return dist(rng);
29  }
30  LD Sigma(LD x) { return 1 / (1 + exp(-x)); }
31
32  LD overall_max_score = 0;
33  for (int main_loop = 0; main_loop < 5; main_loop++) {
34    vector<LD> e(n, (LD)1 / n);
35    for (int tr = 0; tr < 1000; tr++) {
36      vector<LD> ne(n);
37      for (int i = 0; i < n; i++) {
38        ne[i] = Prob();
39      }
```

```
40      LD s = accumulate(all(ne), 0.0L);
41      for (int i = 0; i < n; i++) {
42        ne[i] /= s;
43      }
44      if (eval(ne) > eval(e)) e = ne;
45    }
46    LD t = (LD)0.0002;
47    LD max_score = 0;
48    const LD depr = 0.999995;
49    const int tries = 2E6;
50    const int loop = 1E5;
51
52    LD score_old = eval(e);
53    for (int tr = 0; tr < tries; tr++) {
54  #ifdef LOCAL
55      if (tr % loop == loop - 1) {
56        cout << fixed << setprecision(10) << "current score = " << max_score
57             << ", t = " << t << '\n';
58      }
59  #endif
60      int x = rng() % n, y = rng() % n;
61      if (e[x] < t || x == y) {
62        t *= depr;
63        continue;
64      }
65      e[x] -= t;
66      e[y] += t;
67      LD score_new = eval(e);
68      if (score_new > score_old) {   // ok
69        ;
70      } else {   // revert
71        e[x] += t;
72        e[y] -= t;
73      }
74      score_old = score_new;
75      max_score = max(max_score, score_new);
76      t *= depr;
77    }
78    overall_max_score = max(overall_max_score, max_score);
79  #ifdef LOCAL
80    cout << "Loop #" << main_loop << ": " << max_score << '\n';
81  #endif
82  }
```

8.10   sort.cpp

```cpp
1   void merge_sort(int q[], int l, int r) {
2       if (l >= r) return;
3       int mid = l + r >> 1;
4       merge_sort(q, l, mid);
5       merge_sort(q, mid + 1, r);
6
7       int k = 0, i = l, j = mid + 1;
8       while (i <= mid && j <= r)
9           if (q[i] <= q[j])
10              tmp[k++] = q[i++];
11          else
12              tmp[k++] = q[j++];
13
14      while (i <= mid)
15          tmp[k++] = q[i++];
16      while (j <= r)
17          tmp[k++] = q[j++];
18
19      for (i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];
20  }
21
22  void quick_sort(int q[], int l, int r) {
23      if (l >= r) return;
24      int i = l - 1, j = r + 1, x = q[l + r >> 1];
25      while (i < j) {
26          do i ++ ; while (q[i] < x);
27          do j -- ; while (q[j] > x);
28          if (i < j) swap(q[i], q[j]);
29      }
30      quick_sort(q, l, j), quick_sort(q, j + 1, r);
31  }
32
33  template<class T>
34  void radixsort(T *a, ll n) {
35      int base = 0;
36      rep(i, 1, n) sa[i] = i;
37      rep(k, 1, 5) {
38          rep(i, 0, 255) c[i] = 0;
39          rep(i, 1, n) c[(a[i] >> base) & 255]++;
40          rep(i, 1, 255) c[i] += c[i - 1];
41          per(i, n, 1) {
42              rk[sa[i]] = c[(a[sa[i]] >> base) & 255]--;
43          }
44          rep(i, 1, n) sa[rk[i]] = i;
45          base += 7;
46      }
47  }
```

8.11   高精度.cpp

```cpp
1   vector<int> add(vector<int> &A, vector<int> &B) {
2       if (A.size() < B.size()) return add(B, A);
3       vector<int> C;
4       int t = 0;
5       for (int i = 0; i < A.size(); i ++ ) {
6           t += A[i];
7           if (i < B.size()) t += B[i];
8           C.push_back(t % 10);
9           t /= 10;
10      }
11      if (t) C.push_back(t);
12      return C;
13  }
14
15  vector<int> sub(vector<int> &A, vector<int> &B) {
16      vector<int> C;
17      for (int i = 0, t = 0; i < A.size(); i ++ ) {
18          t = A[i] - t;
19          if (i < B.size()) t -= B[i];
20          C.push_back((t + 10) % 10);
21          if (t < 0) t = 1;
22          else t = 0;
23      }
24      while (C.size() > 1 && C.back() == 0) C.pop_back();
25      return C;
26  }
27
28  vector<int> mul(vector<int> &A, int b) {
29      vector<int> C;
30      int t = 0;
31      for (int i = 0; i < A.size() || t; i ++ ) {
32          if (i < A.size()) t += A[i] * b;
33          C.push_back(t % 10);
34          t /= 10;
35      }
```

```
36        while (C.size() > 1 && C.back() == 0) C.pop_back();
37        return C;
38  }
39
40  vector<int> div(vector<int> &A, int b, int &r) {
41        vector<int> C;
42        r = 0;
43        for (int i = A.size() - 1; i >= 0; i -- ) {
44            r = r * 10 + A[i];
45            C.push_back(r / b);
46            r %= b;
47        }
48        reverse(C.begin(), C.end());
49        while (C.size() > 1 && C.back() == 0) C.pop_back();
50        return C;
51  }
```