南京大学 ACM-ICPC 集训队代码模版库

Linux-4.15.0-65-generic-x86_64-with-Ubuntu-18.04-bionic
XeTeX 3.14159265-2.6-0.99998 (TeX Live 2017/Debian)
CPython 2.7.15+
2019-10-21 19:52:22.967409, build 0062

Contents

1 General **3**
1.1 Code library checksum . . . 3
1.2 Makefile . . . 3
1.3 .vimrc . . . 3
1.4 Stack . . . 3
1.5 Template . . . 3

2 Miscellaneous Algorithms **4**
2.1 2-SAT . . . 4
2.2 Matroid Intersection . . . 4

3 String **5**
3.1 Knuth-Morris-Pratt algorithm . . . 5
3.2 Manacher algorithm . . . 6
3.3 Aho-corasick automaton . . . 6
3.4 Trie . . . 7
3.5 Suffix array . . . 7
3.6 Rolling hash . . . 8

4 Math **8**
4.1 Extended Euclidean algorithm and Chinese remainder theorem . . . 8
4.2 Linear basis . . . 9
4.3 Gauss elimination over finite field . . . 9
4.4 Berlekamp-Massey algorithm . . . 10
4.5 Fast Walsh-Hadamard transform . . . 10
4.6 Fast fourier transform . . . 10
4.7 Number theoretic transform . . . 11
4.8 Sieve of Euler . . . 11
4.9 Sieve of Euler (General) . . . 12
4.10 Miller-Rabin primality test . . . 12
4.11 Integer factorization (Pollard's rho) . . . 13
4.12 Adaptive Simpson's Method . . . 13
4.13 Linear Programming (Simplex) . . . 13

5 Graph Theory **14**
5.1 Vertex biconnected components . . . 14
5.2 Cut vertices . . . 15
5.3 Minimum spanning arborescence, faster . . . 15
5.4 Maximum flow (Dinic) . . . 16
5.5 Maximum cardinality bipartite matching (Hungarian) . . . 17
5.6 Maximum matching of general graph (Edmond's blossom) . . . 17
5.7 Minimum cost maximum flow . . . 18
5.8 Fast LCA . . . 19
5.9 Heavy-light decomposition . . . 20
5.10 Centroid decomposition . . . 20
5.11 DSU on tree . . . 21

6 Data Structures **22**
6.1 Fenwick tree (point update range query) . . . 22
6.2 Fenwick tree (range update point query) . . . 22
6.3 Segment tree . . . 22
6.4 Mo's algorithm . . . 23
6.5 Mo's algorithm on tree . . . 24
6.6 Treap . . . 24
6.7 Link/cut tree . . . 26
6.8 Balanced binary search tree from `pb_ds` . . . 26
6.9 Persistent segment tree, range k-th query . . . 27
6.10 Block list . . . 27
6.11 Persistent block list . . . 29
6.12 Sparse table, range minimum query . . . 30

7 Geometrics **30**
7.1 2D geometric template . . . 30

8 Appendices **32**
8.1 Primes . . . 32
8.1.1 First primes . . . 32
8.1.2 Arbitrary length primes . . . 32
8.1.3 $\sim 1 \times 10^9$ . . . 32
8.1.4 $\sim 1 \times 10^{18}$ . . . 32
8.2 Pell's equation . . . 32
8.3 Burnside's lemma and Polya's enumeration theorem . . . 33
8.4 Supnick TSP . . . 33
8.5 Lagrange's interpolation . . . 33

1 General

1.1 Code library checksum

```
ab14 #!/usr/bin/python3
c502 import re, sys, hashlib
427e
f7db for line in sys.stdin.read().strip().split("\n") :
ddf5     print(hashlib.md5(re.sub(r'\s|//.*', '', line).encode('utf8')).hexdigest()
           [-4:], line)
```

1.2 Makefile

```
dab2 .PHONY : run
427e
207e $(t) : $(t).cpp
2d16   g++ --std=c++14 -Wall -D__LOCAL_DEBUG__ -fsanitize=undefined -fsanitize=
         address -ggdb -pipe -o $@ $<
427e
5f25 run : $(t)
bf3e   ./$(t) < $(t).in
```

1.3 .vimrc

```
914c set nocompatible
733d syntax on
6bbc colorscheme slate
7db5 set number
b0e3 set cursorline
061b set shiftwidth=2
8011 set softtabstop=2
a66d set tabstop=2
d23a set expandtab
5245 set magic
740c set smartindent
bee8 set backspace=indent,eol,start
815d set cmdheight=1
0a40 set laststatus=2
1c67 set whichwrap=b,s,<,>,[,]
```

1.4 Stack

```
const int STK_SZ = 2000000;   bebe
char STK[STK_SZ * sizeof(void*)];   effc
void *STK_BAK;   4e99
   427e
#if defined(__i386__)   7bc9
#define SP "%%esp"   0894
#elif defined(__x86_64__)   ac7a
#define SP "%%rsp"   a9ea
#endif   1937
   427e
int main() {   3117
  asm volatile("mov␣" SP ",%0;␣mov␣%1," SP: "=g"(STK_BAK):"g"(STK+sizeof(STK)):)   3750
    ;
   427e
  // main program   427e
   427e
  asm volatile("mov␣%0," SP::"g"(STK_BAK));   6856
  return 0;   7021
}   95cf
```

1.5 Template

```
#include <bits/stdc++.h>   302f
using namespace std;   421c
   427e
#ifdef __LOCAL_DEBUG__   426f
# define _debug(fmt, ...) fprintf(stderr, "[%s]␣" fmt "\n", \   3341
    __func__, ##__VA_ARGS__)   611f
#else   a8cb
# define _debug(...) ((void) 0)   e6b5
#endif   1937
#define rep(i, n) for (int i=0; i<(n); i++)   0d6c
#define Rep(i, n) for (int i=1; i<=(n); i++)   cfe3
#define range(x) begin(x), end(x)   3505
typedef long long LL;   5cad
typedef unsigned long long ULL;   b773
```

2 Miscellaneous Algorithms

2.1 2-SAT

Usage:

`init(n)`	Initialize the solver with n variables.
`add_clause(x, xval, y, yval)`	Add a clause `(x == xval)-> (y == yval)`.
`solve()`	Solve the problem. Return **`true`** if SAT, or **`false`** if UNSAT.
`operator``[](i)`	Get the value of i-th variable.

```
0f42 const int MAXN = 100005;
03a9 struct twoSAT {
5c83     int n;
8f72     vector<int> G[MAXN*2];
d060     bool mark[MAXN*2];
b42d     int S[MAXN*2], c;
427e
d34f     void init(int n) {
b985         this->n = n;
f9ec         for (int i=0; i < n*2; i++) G[i].clear();
0609         memset(mark, 0, sizeof(mark));
95cf     }
427e
3bd5     bool dfs(int x) {
bd70         if (mark[x^1]) return false;
c96a         if (mark[x]) return true;
fd23         mark[x] = true;
4bea         S[c++] = x;
bd55         for (int u : G[x]) if (!dfs(u)) return false;
3361         return true;
95cf     }
427e
5894     void add_clause(int x, bool xval, int y, bool yval) {
6afe         x = x * 2 + xval;
e680         y = y * 2 + yval;
81cc         G[x^1].push_back(y);
95cf     }
427e
d0cb     bool solve() {
7c39         for (int i=0; i<n*2; i+=2) {
e63f             if (!mark[i] && !mark[i+1]) {
88fb                 c = 0;
                 if (!dfs(i)) {                              f4b9
                     while (c > 0) mark[S[--c]] = false;     3f03
                     if (!dfs(i+1)) return false;            86c5
                 }                                           95cf
             }                                               95cf
         }                                                   95cf
         return true;                                        3361
     }                                                       95cf
                                                             427e
     bool operator[] (int x) { return mark[2*x+1]; }         fb3b
 };                                                          329b
```

2.2 Matroid Intersection

Find the maximum cardinality common independent set of two matroids. Matroids are given by independence oracle.

Usage:

`MatroidOracle`	The independence oracle maintaining an independent set. **Note** that the default constructor must properly initialize inner state to an empty set.
`insert(x)`	Insert element labeled x to the independent set.
`test(x)`	Test whether the set is still independent if x is inserted.
`MatroidIntersection<MT1, MT2>(n)`	Construct the matroid intersection solver with n elements labeled from 0 and matroid oracles `MT1` and `MT2`.
`run()`	Run the algorithm and return the matroid intersection.

```
struct MatroidOracle {                               0935
    MatroidOracle() { /* TODO */ }                   297b
    void insert(int x) { /* TODO */ }                53e5
    bool test(int x) const { /* TODO */ }            ff18
};                                                   329b
                                                     427e
const int MAXN = 8192;                               a015
template <typename MT1, typename MT2>                94cc
struct MatroidIntersection {                         3288
    int n;                                           5c83
    bool in[MAXN] = {}, t[MAXN], vis[MAXN];          5550
    int pre[MAXN];                                   fe84
    vector<int> adj[MAXN];                           0b32
    queue<int> q;                                    93d2
                                                     427e
    MatroidIntersection(int n) : n(n) { }            c152
```

```
427e
2ed1     vector<int> getcur() {
995a         vector<int> ret;
a585         rep (i, n) if (in[i]) ret.push_back(i);
ee0f         return ret;
95cf     }
427e
ca2b     void enqueue(int x, int p) {
e5da         if (vis[x]) return;
f4a6         vis[x] = true; pre[x] = p; q.push(x);
ff59         if (t[x]) throw x;
329b     };
427e
9081     vector<int> run() {
1026         while (true) {
c40f             vector<int> cur = getcur();
6f47             fill(vis, vis + n, 0);
943b             rep (i, n) adj[i].clear();
0e02             MT2 mt2;
3e54             for (int i : cur) mt2.insert(i);
191d             rep (i, n) t[i] = mt2.test(i);
e167             vector<MT1> mt1s(cur.size());
46d2             vector<MT2> mt2s(cur.size());
660b             rep (i, cur.size()) rep (j, cur.size()) if (i != j) {
3cd7                 mt1s[i].insert(cur[j]);
9680                 mt2s[i].insert(cur[j]);
95cf             }
e8d7             rep (i, n) if (!in[i]) rep (j, cur.size()) {
3fe9                 if (mt1s[j].test(i)) adj[cur[j]].push_back(i);
645e                 if (mt2s[j].test(i)) adj[i].push_back(cur[j]);
95cf             }
cf76             q = {};
85eb             try {
2f4f                 MT1 mt1;
2f34                 for (int i : cur) mt1.insert(i);
4053                 rep (i, n) if (mt1.test(i)) enqueue(i, -1);
1c7d                 while (q.size()) {
c048                     int u = q.front(); q.pop();
a697                     for (int v : adj[u]) enqueue(v, u);
95cf                 }
5a9a             } catch (int v) {
a8f3                 while (v >= 0) { in[v] ^= 1; v = pre[v]; }
b333                 continue;
95cf             }
```

```
                break;                                                6173
            };                                                        329b
            return getcur();                                          f2de
        }                                                             95cf
};                                                                    329b
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
const int SIZE = 10005;                                               2836
                                                                      427e
struct kmp_matcher {                                                  d02b
  char p[SIZE];                                                       2d81
  int fail[SIZE];                                                     9847
  int len;                                                            57b7
                                                                      427e
  void construct(const char* needle) {                                60cf
    len = strlen(p);                                                  aaa1
    strcpy(p, needle);                                                3a87
    fail[0] = fail[1] = 0;                                            3dd4
    for (int i = 1; i < len; i++) {                                   d8a8
      int j = fail[i];                                                147f
      while (j && p[i] != p[j]) j = fail[j];                          3c79
      fail[i + 1] = p[i] == p[j] ? j + 1 : 0;                         4643
    }                                                                 95cf
  }                                                                   95cf
                                                                      427e
  inline void found(int pos) {                                        c464
    // ! add codes for having found at pos                            427e
  }                                                                   95cf
                                                                      427e
  void match(const char* haystack) {  // must be called after construct  2daf
    const char* t = haystack;                                         700f
    int n = strlen(t);                                                8482
    int j = 0;                                                        8fd0
    rep(i, n) {                                                       be8e
      while (j && p[j] != t[i]) j = fail[j];                          4e19
      if (p[j] == t[i]) j++;                                          b5d5
      if (j == len) found(i - len + 1);                               f024
    }                                                                 95cf
```

```
95cf   }
329b };
```

3.2 Manacher algorithm

```
81d4 struct Manacher {
cd09   int Len;
9255   vector<int> lc;
b301   string s;
427e
ec07   void work() {
c033     lc[1] = 1;
6bef     int k = 1;
427e
491f     for (int i = 2; i <= Len; i++) {
7957       int p = k + lc[k] - 1;
5e04       if (i <= p) {
24a1         lc[i] = min(lc[2 * k - i], p - i + 1);
8e2e       } else {
e0e5         lc[i] = 1;
95cf       }
74ff       while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
2b9a       if (i + lc[i] > k + lc[k]) k = i;
95cf     }
95cf   }
427e
bfd5   void init(const char *tt) {
aaaf     int len = strlen(tt);
f701     s.resize(len * 2 + 10);
7045     lc.resize(len * 2 + 10);
8e13     s[0] = '*';
ae54     s[1] = '#';
1321     for (int i = 0; i < len; i++) {
e995       s[i * 2 + 2] = tt[i];
69fd       s[i * 2 + 1] = '#';
95cf     }
43fd     s[len * 2 + 1] = '#';
75d1     s[len * 2 + 2] = '\0';
61f7     Len = len * 2 + 2;
3e7a     work();
95cf   }
427e
```

```
  pair<int, int> maxpal(int l, int r) {                  b194
    int center = l + r + 1;                              901a
    int rad = lc[center] / 2;                            ffb2
    int rmid = (l + r + 1) / 2;                          ab54
    int rl = rmid - rad, rr = rmid + rad - 1;            17e4
    if ((r ^ l) & 1) {                                   3908
    } else rr++;                                         69f3
    return {max(l, rl), min(r, rr)};                     69dc
  }                                                      95cf
};                                                       329b
```

3.3 Aho-corasick automaton

```
struct AC : Trie {                                       a1ad
  int fail[MAXN];                                        9143
  int last[MAXN];                                        daca
                                                         427e
  void construct() {                                     8690
    queue<int> q;                                        93d2
    fail[0] = 0;                                         a7a6
    rep(c, CHARN) {                                      ce3c
      if (int u = tr[0][c]) {                            b1c6
        fail[u] = 0;                                     a506
        q.push(u);                                       3e14
        last[u] = 0;                                     f689
      }                                                  95cf
    }                                                    95cf
    while (!q.empty()) {                                 cc78
      int r = q.front();                                 31f0
      q.pop();                                           15dd
      rep(c, CHARN) {                                    ce3c
        int u = tr[r][c];                                ab59
        if (!u) {                                        0ef5
          tr[r][c] = tr[fail[r]][c];                     9d58
          continue;                                      b333
        }                                                95cf
        q.push(u);                                       3e14
        int v = fail[r];                                 b3ff
        while (v && !tr[v][c]) v = fail[v];              d2ea
        fail[u] = tr[v][c];                              c275
        last[u] = tag[fail[u]] ? fail[u] : last[fail[u]]; 654c
      }                                                  95cf
```

```
    }
  }

  void found(int pos, int j) {
    if (j) {
      // ! add codes for having found word with tag[j]
      found(pos, last[j]);
    }
  }

  void find(const char* text) {  // must be called after construct()
    int p = 0, c, len = strlen(text);
    rep(i, len) {
      c = id(text[i]);
      p = tr[p][c];
      if (tag[p])
        found(i, p);
      else if (last[p])
        found(i, last[p]);
    }
  }
};
```

3.4 Trie

```
const int MAXN = 12000;
const int CHARN = 26;

inline int id(char c) { return c - 'a'; }

struct Trie {
  int n;
  int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail
  int tag[MAXN];

  Trie() {
    memset(tr[0], 0, sizeof(tr[0]));
    tag[0] = 0;
    n = 1;
  }

  // tag should not be 0
  void add(const char* s, int t) {
    int p = 0, c, len = strlen(s);
    rep(i, len) {
      c = id(s[i]);
      if (!tr[p][c]) {
        memset(tr[n], 0, sizeof(tr[n]));
        tag[n] = 0;
        tr[p][c] = n++;
      }
      p = tr[p][c];
    }
    tag[p] = t;
  }

  // returns 0 if not found
  // AC automaton does not need this function
  int search(const char* s) {
    int p = 0, c, len = strlen(s);
    rep(i, len) {
      c = id(s[i]);
      if (!tr[p][c]) return 0;
      p = tr[p][c];
    }
    return tag[p];
  }
};
```

3.5 Suffix array

The character immediately after the end of the string **MUST** be set to the **UNIQUE SMALLEST** element.

Usage:

`s[]`	the source string
`sa[i]`	the index of starting position of i-th suffix
`rk[i]`	the number of suffixes less than the suffix starting from i
`h[i]`	the longest common prefix between the i-th and $(i-1)$-th lexicographically smallest suffixes
`n`	size of source string
`m`	size of character set

```
void radix_sort(int x[], int y[], int sa[], int n, int m) {
    static int cnt[1000005];    // size > max(n, m)
    fill(cnt, cnt + m, 0);
```

```
93b7      rep (i, n) cnt[x[y[i]]]++;
9154      partial_sum(cnt, cnt + m, cnt);
acac      for (int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
95cf  }
427e
c939  void suffix_array(int s[], int sa[], int rk[], int n, int m) {
a69a      static int y[1000005];  // size > n
7306      copy(s, s + n, rk);
afbb      iota(y, y + n, 0);
7b42      radix_sort(rk, y, sa, n, m);
c8c2      for (int j = 1, p = 0; j <= n; j <<= 1, m = p, p = 0) {
8c3a          for (int i = n - j; i < n; i++) y[p++] = i;
9323          rep (i, n) if (sa[i] >= j) y[p++] = sa[i] - j;
9e9d          radix_sort(rk, y, sa, n, m + 1);
ae41          swap_ranges(rk, rk + n, y);
ffd2          rk[sa[0]] = p = 1;
445e          for (int i = 1; i < n; i++)
f8dc              rk[sa[i]] = ((y[sa[i]] == y[sa[i-1]] and y[sa[i]+j] == y[sa[i-1]+j])
                      ? p : ++p);
02f0          if (p == n) break;
95cf      }
97d9      rep (i, n) rk[sa[i]] = i;
95cf  }
427e
1715  void calc_height(int s[], int sa[], int rk[], int h[], int n) {
c41f      int k = 0;
f313      h[0] = 0;
be8e      rep (i, n) {
0883          k = max(k - 1, 0);
527d          if (rk[i]) while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;
56b7          h[rk[i]] = k;
95cf      }
95cf  }
```

3.6 Rolling hash

PLEASE call `init_hash()` in **`int`** `main()`!
Usage:

`build(str)`	Construct the hasher with given string.
`operator``()(l, r)`	Get hash value of substring $[l, r)$.

```
1e42  const LL mod = 1006658951440146419, g = 967;
9f60  const int MAXN = 200005;
LL pg[MAXN];                                                                        0291
                                                                                    427e
inline LL mul(LL x, LL y) { return __int128_t(x) * y % mod; }                      dfe7
                                                                                    427e
void init_hash() {   // must be called in `int main()`                              599a
    pg[0] = 1;                                                                      286f
    for (int i = 1; i < MAXN; i++) pg[i] = mul(pg[i-1], g);                         4af8
}                                                                                   95cf
                                                                                    427e
struct hasher {                                                                     7e62
    LL val[MAXN];                                                                   534a
                                                                                    427e
    void build(const char *str) {   // assume lower-case letter only                4554
        for (int i = 0; str[i]; i++)                                                f937
            val[i+1] = (mul(val[i], g) + str[i]) % mod;                             9645
    }                                                                               95cf
                                                                                    427e
    LL operator() (int l, int r) { // [l, r)                                        19f8
        return (val[r] - mul(val[l], pg[r-l]) + mod) % mod;                         9986
    }                                                                               95cf
};                                                                                  329b
```

4 Math

4.1 Extended Euclidean algorithm and Chinese remainder theorem

```
void exgcd(LL a, LL b, LL &g, LL &x, LL &y) {         4fba
    if (!b) g = a, x = 1, y = 0;                      7db6
    else {                                            037f
        exgcd(b, a % b, g, y, x);                     ffca
        y -= x * (a / b);                             d798
    }                                                 95cf
}                                                     95cf
                                                      427e
LL crt(LL r[], LL p[], int n) {                       e491
    LL q = 1, ret = 0;                                84e6
    rep (i, n) q *= p[i];                             00d9
    rep (i, n) {                                      be8e
        LL m = q / p[i];                              98b4
        LL d, x, y;                                   9f4f
        exgcd(p[i], m, d, x, y);                      b082
```

```
        ret = (ret + y * m * r[i]) % q;
    }
    return (q + ret) % q;
}
```

4.2 Linear basis

```
const int MAXD = 30;
struct linearbasis {
    ULL b[MAXD] = {};

    bool insert(LL v) {
        for (int j = MAXD - 1; j >= 0; j--) {
            if (!(v & (1ll << j))) continue;
            if (b[j]) v ^= b[j]
            else {
                for (int k = 0; k < j; k++)
                    if (v & (1ll << k)) v ^= b[k];
                for (int k = j + 1; k < MAXD; k++)
                    if (b[k] & (1ll << j)) b[k] ^= v;
                b[j] = v;
                return true;
            }
        }
        return false;
    }
};
```

4.3 Gauss elimination over finite field

```
const LL p = 1000000007;

LL powmod(LL b, LL e) {
  LL r = 1;
  while (e) {
    if (e & 1) r = r * b % p;
    b = b * b % p;
    e >>= 1;
  }
  return r;
}

typedef vector<LL> VLL;
typedef vector<VLL> VVLL;

LL gauss(VVLL &a, VVLL &b) {
  const int n = a.size(), m = b[0].size();
  vector<int> irow(n), icol(n), ipiv(n);
  LL det = 1;

  rep (i, n) {
    int pj = -1, pk = -1;
    rep (j, n) if (!ipiv[j])
      rep (k, n) if (!ipiv[k])
        if (pj == -1 || a[j][k] > a[pj][pk]) {
          pj = j;
          pk = k;
        }
    if (a[pj][pk] == 0) return 0;
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det = (p - det) % p;
    irow[i] = pj;
    icol[i] = pk;

    LL c = powmod(a[pk][pk], p - 2);
    det = det * a[pk][pk] % p;
    a[pk][pk] = 1;
    rep (j, n) a[pk][j] = a[pk][j] * c % p;
    rep (j, m) b[pk][j] = b[pk][j] * c % p;
    rep (j, n) if (j != pk) {
      c = a[j][pk];
      a[j][pk] = 0;
      rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
      rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
    }
  }

  for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
  }
  return det;
}
```

4.4 Berlekamp-Massey algorithm

Call `berlekamp()` with input sequence $(x_0, x_1, \cdots, x_{n-1})$. Return a vector of coefficients $(c_0 = 1, c_1, \cdots, c_{m-1})$ with minimum m, such that $\sum_{i=0}^{m} c_i x_{j-i} = 0$ for all possible j.

```
6e50 LL mod = 1000000007;
97db vector<LL> berlekamp(const vector<LL>& a) {
8904     vector<LL> p = {1}, r = {1};
075b     LL dif = 1;
8bc9     rep (i, a.size()) {
1b35         LL u = 0;
bd0b         rep (j, p.size()) u = (u + p[j] * a[i-j]) % mod;
eae9         if (u == 0) {
b14c             r.insert(r.begin(), 0);
8e2e         } else {
0c78             auto op = p;
02f6             p.resize(max(p.size(), r.size() + 1));
0a2e             LL idif = powmod(dif, mod - 2);
9b57             rep (j, r.size())
dacc                 p[j+1] = (p[j+1] - r[j] * idif % mod * u % mod + mod) % mod;
bcd1             dif = u; r = op;
95cf         }
95cf     }
e149     return p;
95cf }
```

4.5 Fast Walsh-Hadamard transform

```
061e void fwt(int* a, int n){
5595     for (int d = 1; d < n; d <<= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = x+y, a[i+j+d] = x-y;    // xor
427e                 // a[i+j] = x+y;                    // and
427e                 // a[i+j+d] = x+y;                  // or
95cf             }
95cf }
427e
4db1 void ifwt(int* a, int n){
5595     for (int d = 1; d < n; d <<= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
                     int x = a[i+j], y = a[i+j+d];                          7796
                     // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor     427e
                     // a[i+j] = x-y;                            // and     427e
                     // a[i+j+d] = y-x;                          // or      427e
                 }                                                          95cf
}                                                                           95cf
                                                                            427e
void conv(int* a, int* b, int n){                                           2ab6
    fwt(a, n);                                                              950a
    fwt(b, n);                                                              e427
    rep(i, n) a[i] *= b[i];                                                 8a42
    ifwt(a, n);                                                             430f
}                                                                           95cf
```

4.6 Fast fourier transform

```
const int NMAX = 1<<20;                                                     4e09
                                                                            427e
typedef complex<double> cplx;                                               3fbf
                                                                            427e
const double PI = 2*acos(0.0);                                              abd1
struct FFT{                                                                 12af
    int rev[NMAX];                                                          c47c
    cplx omega[NMAX], oinv[NMAX];                                           27d7
    int K, N;                                                               9827
                                                                            427e
    FFT(int k){                                                             1442
        K = k; N = 1 << k;                                                  e209
        rep (i, N){                                                         b393
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));                       7ba3
            omega[i] = polar(1.0, 2.0 * PI / N * i);                        1908
            oinv[i] = conj(omega[i]);                                       a166
        }                                                                   95cf
    }                                                                       95cf
                                                                            427e
    void dft(cplx* a, cplx* w){                                             b941
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);                   a215
        for (int l = 2; l <= N; l *= 2){                                    ac6e
            int m = l/2;                                                    2969
            for (cplx* p = a; p != a + N; p += l)                           b3cf
                rep (k, m){                                                 c24f
                    cplx t = w[N/l*k] * p[k+m];                             fe06
```

```
ecbf                     p[k+m] = p[k] - t; p[k] += t;
95cf                 }
95cf             }
95cf         }
427e
617b     void fft(cplx* a){dft(a, omega);}
a123     void ifft(cplx* a){
3b2f         dft(a, oinv);
57fc         rep (i, N) a[i] /= N;
95cf     }
427e
bdc0     void conv(cplx* a, cplx* b){
6497         fft(a); fft(b);
12a5         rep (i, N) a[i] *= b[i];
f84e         ifft(a);
95cf     }
329b };
```

4.7 Number theoretic transform

```
4ab9 const int NMAX = 1<<21;
427e
427e // 998244353 = 7*17*2^23+1, G = 3
fb9a const int P = 1004535809, G = 3; // = 479*2^21+1
427e
87ab struct NTT{
c47c     int rev[NMAX];
0eda     LL omega[NMAX], oinv[NMAX];
81af     int g, g_inv; // g: g_n = G^((P-1)/n)
9827     int K, N;
427e
2a2c     LL powmod(LL b, LL e){
95a2         LL r = 1;
3e90         while (e){
6624             if (e&1) r = r * b % P;
489e             b = b * b % P;
16fc             e >>= 1;
95cf         }
547e         return r;
95cf     }
427e
f420     NTT(int k){
        K = k; N = 1 << k;                                   e209
        g = powmod(G, (P-1)/N);                              7652
        g_inv = powmod(g, N-1);                              4b3a
        omega[0] = oinv[0] = 1;                              e04f
        rep (i, N){                                          b393
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));        7ba3
            if (i){                                          ad4f
                omega[i] = omega[i-1] * g % P;               8d8b
                oinv[i] = oinv[i-1] * g_inv % P;             9e14
            }                                                95cf
        }                                                    95cf
    }                                                        95cf
                                                             427e
    void _ntt(LL* a, LL* w){                                 9668
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);    a215
        for (int l = 2; l <= N; l *= 2){                     ac6e
            int m = l/2;                                     2969
            for (LL* p = a; p != a + N; p += l)              7a1d
                rep (k, m){                                  c24f
                    LL t = w[N/l*k] * p[k+m] % P;            0ad3
                    p[k+m] = (p[k] - t + P) % P;             6209
                    p[k] = (p[k] + t) % P;                   fa1b
                }                                            95cf
        }                                                    95cf
    }                                                        95cf
                                                             427e
    void ntt(LL* a){_ntt(a, omega);}                         92ea
    void intt(LL* a){                                        5daf
        LL inv = powmod(N, P-2);                             1f2a
        _ntt(a, oinv);                                       9910
        rep (i, N) a[i] = a[i] * inv % P;                    a873
    }                                                        95cf
                                                             427e
    void conv(LL* a, LL* b){                                 3a5b
        ntt(a); ntt(b);                                      ad16
        rep (i, N) a[i] = a[i] * b[i] % P;                   e49e
        intt(a);                                             5748
    }                                                        95cf
};                                                           329b
```

4.8 Sieve of Euler

```
cfc3 const int MAXX = 1e7+5;
5861 bool p[MAXX];
73ae int prime[MAXX], sz;
427e
9bc6 void sieve(){
9628     p[0] = p[1] = 1;
1ec8     for (int i = 2; i < MAXX; i++){
bf28         if (!p[i]) prime[sz++] = i;
e82c         for (int j = 0; j < sz && i*prime[j] < MAXX; j++){
b6a9             p[i*prime[j]] = 1;
5f51             if (i % prime[j] == 0) break;
95cf         }
95cf     }
95cf }
```

4.9 Sieve of Euler (General)

```
b62e namespace sieve {
6589   constexpr int MAXN = 10000007;
e982   bool p[MAXN]; // true if not prime
6ae8   int prime[MAXN], sz;
cbf7   int pval[MAXN], pcnt[MAXN];
6030   int f[MAXN];
427e
76f6   void exec(int N = MAXN) {
9628     p[0] = p[1] = 1;
427e
8a8a     pval[1] = 1;
bdda     pcnt[1] = 0;
c6b9     f[1] = 1;
427e
a643     for (int i = 2; i < N; i++) {
01d6       if (!p[i]) {
b2b2         prime[sz++] = i;
37d9         for (LL j = i; j < N; j *= i) {
758c           int b = j / i;
81fd           pval[j] = i * pval[b];
e0f3           pcnt[j] = pcnt[b] + 1;
a96c           f[j] = ______; // f[j] = f(i^pcnt[j])
95cf         }
95cf       }
34c0       for (int j = 0; i * prime[j] < N; j++) {
        int x = i * prime[j]; p[x] = 1; f87a
        if (i % prime[j] == 0) { 20cc
          pval[x] = pval[i] * prime[j]; 9985
          pcnt[x] = pcnt[i] + 1; 3f93
        } else { 8e2e
          pval[x] = prime[j]; cc91
          pcnt[x] = 1; 6322
        } 95cf
        if (x != pval[x]) { 6191
          f[x] = f[x / pval[x]] * f[pval[x]] d614
        } 95cf
        if (i % prime[j] == 0) break; 5f51
      } 95cf
    } 95cf
  } 95cf
} 95cf
```

4.10 Miller-Rabin primality test

The array a[] (excluding senitel, i.e. LLONG_MAX) should be

{2}	when $n < 2,047$.
{2, 7, 61}	when $n < 4,759,123,141$ (2^{32}) .
{2, 3, 5, 7, 11}	when $n < 2.1 \times 10^{12}$.
{2, 325, 9375, 28178, 450775, 9780504, 1795265022}	when $n < 2^{64}$.

```
bool test(LL n){ f16f
    if (n < 3) return n==2; 59f2
    // ! The array a[] should be modified if the range of x changes. 427e
    const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX}; 3f11
    LL r = 0, d = n-1, x; c320
    while (~d & 1) d >>= 1, r++; f410
    for (int i=0; a[i] < n; i++){ 2975
        x = powmod(a[i], d, n); // ! powmod must use for 64bit mulmod ece1
        if (x == 1 || x == n-1) goto next; 7f99
        rep (i, r) { e257
            x = mulmod(x, x, n); d7ff
            if (x == n-1) goto next; 8d2e
        } 95cf
        return false; 438e
next:; d490
    } 95cf
    return true; 3361
```

```
}
```

4.11 Integer factorization (Pollard's rho)

```
ULL gcd(ULL a, ULL b) {return b ? gcd(b, a % b) : a;}

ULL PollardRho(ULL n){
    ULL c, x, y, d = n;
    if (~n&1) return 2;
    while (d == n){
        x = y = 2;
        d = 1;
        c = rand() % (n - 1) + 1;
        while (d == 1){
            x = (mulmod(x, x, n) + c) % n;
            y = (mulmod(y, y, n) + c) % n;
            y = (mulmod(y, y, n) + c) % n;
            d = gcd(x>y ? x-y : y-x, n);
        }
    }
    return d;
}
```

4.12 Adaptive Simpson's Method

The Simpson's formula has order 3 algebraic precision.

Usage:

`integrate(l, r, eps, est, fn)`	Integrate the function `fn` on interval $[l, r]$. `eps` is the estimated precision, while `est` is the current estimation, which can be set to arbitrary value initially.

```
template <typename T>
double simpson(double l, double r, T&& f) {
    double mid = (l + r) / 2;
    return (f(l) + 4 * f(mid) + f(r)) * (r - l) / 6.0;
}

template <typename T>
double integrate(double l, double r, double eps, double est, T&& f) {
    double mid = (l + r) / 2;
    double lv = simpson(l, mid, f), rv = simpson(mid, r, f);
    if (fabs(lv + rv - est) <= 15.0 * eps)
        return lv + rv + (lv + rv - est) / 15.0;
    return integrate(l, mid, eps, lv, f) + integrate(mid, r, eps, rv, f);
}
```

4.13 Linear Programming (Simplex)

This function solves the following linear program

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b \\ & x \ge 0 \end{aligned}$$

If the program is infeasible, `NAN` is returned; if the program is unbounded, `DBL_MAX` is returned; otherwise, the optimal target is returned and the arguments are stored in `x`.

```
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const double EPS = 1e-9;

double LPSolve(VVD A, VD b, VD c, VD& x) {
    int m = b.size(), n = c.size();
    VI B(m), N(n+1);
    VVD D(m+2, VD(n+2));
    rep (i, m) rep (j, n) D[i][j] = A[i][j];
    rep (i, m) { B[i] = n + i; D[i][n] = -1; D[i][n+1] = b[i]; }
    rep (j, n) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;

    auto pivot = [&] (int r, int s) {
        double inv = 1.0 / D[r][s];
        rep (i, m+2) if (i != r) rep (j, n+2) if (j != s)
            D[i][j] -= D[r][j] * D[i][s] * inv;
        rep (j, n+2) if (j != s) D[r][j] *= inv;
        rep (i, m+2) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv; swap(B[r], N[s]);
    };

    auto simplex = [&](int phase) {
        int x = m + (phase == 1);
        while (true) {
```

```
0676            int s = -1;
7e4d            for (int j = 0; j <= n; j++) {
30f5                if (phase == 2 and N[j] == -1) continue;
537c                if (s == -1 or D[x][j] < D[x][s] or
3262                    D[x][j] == D[x][s] and N[j] < N[s]) s = j;
95cf            }
083a            if (s < 0 or D[x][s] > -EPS) return true;
bfc5            int r = -1;
356f            for (int i = 0; i < m; i++) {
691d                if (D[i][s] < EPS) continue;
6855                if (r == -1 or D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] or
26b3                    D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] and
412f                    B[i] < B[r]) r = i;
95cf            }
d829            if (r == -1) return false; else pivot(r, s);
95cf        }
329b    };
427e
7c08    int r = 0;
468b    for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
8257    if (D[r][n+1] <= -EPS) {
d48d        pivot(r, n);
0175        if (!simplex(1) or D[m+1][n+1] < -EPS) return NAN;
fc91        rep (i, m) if (B[i] == -1) {
0676            int s = -1;
1e86            for (int j = 0; j <= n; j++) if (s == -1 or D[i][j] < D[i][s]
a48f                or D[i][j] == D[i][s] and N[j] < N[s]) s = j;
c4cd            pivot(i, s);
95cf        }
95cf    }
e566    if (!simplex(2)) return DBL_MAX;
8720    x = VD(n);
3232    rep (i, m) if (B[i] < n) x[B[i]] = D[i][n+1];
bbe4    return D[m][n+1];
95cf }
```

5 Graph Theory

5.1 Vertex biconnected components

```
0f42 const int MAXN = 100005;
struct graph {                                                          2ea0
    int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;        33ae
    vector<int> adj[MAXN], bcc[MAXN];                                   848f
    set<pair<int, int>> bcce[MAXN];                                     6b06
                                                                        427e
    stack<pair<int, int>> s;                                            76f7
                                                                        427e
    void add_edge(int u, int v) {                                       bfab
        adj[u].push_back(v);                                            c71a
        adj[v].push_back(u);                                            a717
    }                                                                   95cf
                                                                        427e
    int dfs(int u, int fa) {                                            7d3c
        int lowu = pre[u] = ++dfs_clock;                                9fe6
        int child = 0;                                                  ec14
        for (int v : adj[u]) {                                          18f6
            if (!pre[v]) {                                              173e
                s.push({u, v});                                         e7f8
                child++;                                                fdcf
                int lowv = dfs(v, u);                                   f851
                lowu = min(lowu, lowv);                                 189c
                if (lowv >= pre[u]) {                                   b687
                    iscut[u] = 1;                                       6323
                    bcc[bcc_cnt].clear();                               57eb
                    bcce[bcc_cnt].clear();                              90b8
                    while (1) {                                         a147
                        int xu, xv;                                     a6a3
                        tie(xu, xv) = s.top(); s.pop();                 a0c3
                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)}); 0ef5
                        if (bccno[xu] != bcc_cnt) {                     3db2
                            bcc[bcc_cnt].push_back(xu);                 e0db
                            bccno[xu] = bcc_cnt;                        d27f
                        }                                               95cf
                        if (bccno[xv] != bcc_cnt) {                     f357
                            bcc[bcc_cnt].push_back(xv);                 752b
                            bccno[xv] = bcc_cnt;                        57c9
                        }                                               95cf
                        if (xu == u && xv == v) break;                  7096
                    }                                                   95cf
                    bcc_cnt++;                                          03f5
                }                                                       95cf
            } else if (pre[v] < pre[u] && v != fa) {                    7470
                s.push({u, v});                                         e7f8
                lowu = min(lowu, pre[v]);                               f115
```

```
            }
        }
        if (fa < 0 && child == 1) iscut[u] = 0;
        return lowu;
    }

    void find_bcc(int n) {
        memset(pre, 0, sizeof pre);
        memset(iscut, 0, sizeof iscut);
        memset(bccno, -1, sizeof bccno);
        dfs_clock = bcc_cnt = 0;
        rep (i, n) if (!pre[i]) dfs(i, -1);
    }
};
```

5.2 Cut vertices

If the graph is unconnected, the algorithm should be run on each component. One may run `Rep (i, n)if (!dfn[i])tarjan(i, i)` for unconnected graph.

Usage:

`add_edge(u, v)`	Add an undirected edge (u, v).
`tarjan(u, fa)`	Run Tarjan's algorithm on tree rooted at `fa`. Please call with identical `u` and `fa`.
`cut[v]`	Whether v is a cut vertex.

```
const int MAXN = 200005;
vector<int> adj[MAXN];
int dfn[MAXN], low[MAXN], idx;
bool cut[MAXN];

void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}

void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++idx;
    int child = 0;
    for (int v : adj[u]) {
        if (!dfn[v]) {
            tarjan(v, fa); low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u] && u != fa) cut[u] = true;
            child += u == fa;
        }
        low[u] = min(low[u], dfn[v]);
    }
    if (u == fa && child > 1) cut[u] = true;
}
```

5.3 Minimum spanning arborescence, faster

All vertices are 1-based. Clear the fields when reuse the struct.

Usage:

`add_edge(u, v, w)`	Add an edge from u to v with weight w.
`run(n, rt)`	Compute the total weight of MSA rooted at `rt`. If not exist, retun `LLONG_MIN`.

Time Complexity: $O(|E| \log^2 |V|)$

```
const int MAXN = 300005;
typedef pair<LL, int> pii;
struct MDST {
    priority_queue<pii, vector<pii>, greater<pii>> heap[MAXN];
    LL shift[MAXN];
    int fa[MAXN], vis[MAXN];

    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }

    void unite(int x, int y) {
        x = find(x); y = find(y); fa[y] = x; if (x == y) return;
        if (heap[x].size() < heap[y].size()) {
            swap(heap[x], heap[y]);
            swap(shift[x], shift[y]);
        }
        while (heap[y].size()) {
            auto p = heap[y].top(); heap[y].pop();
            heap[x].emplace(p.first - shift[y] + shift[x], p.second);
        }
    }

    void add_edge(int u, int v, LL w) { heap[v].emplace(w, u); }

    LL run(int n, int rt) {
        LL ans = 0;
        iota(fa, fa + n + 1, 0);
        Rep (i, n) if (find(i) != find(rt)) {
```

```
a7b1             int u = find(i);
010e             stack<int, vector<int>> s;
eff5             while (find(u) != find(rt)) {
0dda                 if (vis[u]) while (s.top() != u) {
c593                     vis[s.top()] = 0; unite(u, s.top()); s.pop();
83c4                 } else { vis[u] = 1; s.push(u); }
c76e                 while (heap[u].size()) {
b385                     ans += heap[u].top().first - shift[u];
dde2                     shift[u] = heap[u].top().first;
da47                     if (find(heap[u].top().second) != u) break;
9fbb                     heap[u].pop();
95cf                 }
6961                 if (heap[u].empty()) return LLONG_MIN;
87e6                 u = find(heap[u].top().second);
95cf             }
2d46             while (s.size()) { vis[s.top()] = 0; unite(rt, s.top()); s.pop(); }
95cf         }
4206         return ans;
95cf     }
329b };
```

5.4 Maximum flow (Dinic)

Usage:

`add_edge(u, v, c)`	Add an edge from u to v with capacity c.
`max_flow(s, t)`	Compute maximum flow from s to t.

Time Complexity: For general graph, $O(V^2E)$; for network with unit capacity, $O(\min\{V^{2/3}, \sqrt{E}\}E)$; for bipartite network, $O(\sqrt{V}E)$.

```
bcf8 struct edge{
60e2     int from, to;
5e6d     LL cap, flow;
329b };
427e
e2cd const int MAXN = 1005;
9062 struct Dinic {
4dbf     int n, m, s, t;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
bbb6     bool vis[MAXN];
b40a     int d[MAXN];
ddec     int cur[MAXN];
427e
    void add_edge(int from, int to, LL cap) {                          5973
        edges.push_back(edge{from, to, cap, 0});                       7b55
        edges.push_back(edge{to, from, 0, 0});                         1db7
        m = edges.size();                                              fe77
        G[from].push_back(m-2);                                        dff5
        G[to].push_back(m-1);                                          8f2d
    }                                                                  95cf
                                                                       427e
    bool bfs() {                                                       1836
        memset(vis, 0, sizeof(vis));                                   3b73
        queue<int> q;                                                  93d2
        q.push(s);                                                     5d13
        vis[s] = 1;                                                    2cd2
        d[s] = 0;                                                      721d
        while (!q.empty()) {                                           cc78
            int x = q.front(); q.pop();                                66ba
            for (int i = 0; i < G[x].size(); i++) {                    3b61
                edge& e = edges[G[x][i]];                              b510
                if (!vis[e.to] && e.cap > e.flow) {                    bba9
                    vis[e.to] = 1;                                     cd72
                    d[e.to] = d[x] + 1;                                cf26
                    q.push(e.to);                                      ca93
                }                                                      95cf
            }                                                          95cf
        }                                                              95cf
        return vis[t];                                                 b23b
    }                                                                  95cf
                                                                       427e
    LL dfs(int x, LL a) {                                              9252
        if (x == t || a == 0) return a;                                6904
        LL flow = 0, f;                                                8bf9
        for (int& i = cur[x]; i < G[x].size(); i++) {                  f515
            edge& e = edges[G[x][i]];                                  b510
            if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0) 2374
              {
                e.flow += f;                                           1cce
                edges[G[x][i]^1].flow -= f;                            e16d
                flow += f;                                             a74d
                a -= f;                                                23e5
                if(a == 0) break;                                      97ed
            }                                                          95cf
        }                                                              95cf
        return flow;                                                   84fb
    }                                                                  95cf
```

```
427e
5bf2     LL max_flow(int s, int t) {
590d         this->s = s; this->t = t;
62e2         LL flow = 0;
ed58         while (bfs()) {
f326             memset(cur, 0, sizeof(cur));
fb3a             flow += dfs(s, LLONG_MAX);
95cf         }
84fb         return flow;
95cf     }
427e
c72e     vector<int> min_cut() { // call this after maxflow
1df9         vector<int> ans;
df9a         for (int i = 0; i < edges.size(); i++) {
56d8             edge& e = edges[i];
46a2             if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);
95cf         }
4206         return ans;
95cf     }
329b };
```

5.5 Maximum cardinality bipartite matching (Hungarian)

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
0d6c #define rep(i, n) for (int i = 0; i < (n); i++)
cfe3 #define Rep(i, n) for (int i = 1; i <= (n); i++)
8843 #define range(x) (x).begin(), (x).end()
5cad typedef long long LL;
427e
84ee struct Hungarian{
fbf6     int nx, ny;
9ec6     vector<int> mx, my;
9d4c     vector<vector<int> > e;
edec     vector<bool> mark;
427e
8324     void init(int nx, int ny){
c1d1         this->nx = nx;
f9c1         this->ny = ny;
ac92         mx.resize(nx); my.resize(ny);
3f11         e.clear(); e.resize(nx);
        mark.resize(nx);                         1023
    }                                            95cf
                                                 427e
    inline void add(int a, int b){               4589
        e[a].push_back(b);                       486c
    }                                            95cf
                                                 427e
    bool augment(int i){                         0c2b
        if (!mark[i]) {                          207c
            mark[i] = true;                      dae4
            for (int j : e[i]){                  6a1e
                if (my[j] == -1 || augment(my[j])){   0892
                    mx[i] = j; my[j] = i;        9ca3
                    return true;                 3361
                }                                95cf
            }                                    95cf
        }                                        95cf
        return false;                            438e
    }                                            95cf
                                                 427e
    int match(){                                 3fac
        int ret = 0;                             5b57
        fill(range(mx), -1);                     b0f1
        fill(range(my), -1);                     b957
        rep (i, nx){                             4ed1
            fill(range(mark), false);            13a5
            if (augment(i)) ret++;               cc89
        }                                        95cf
        return ret;                              ee0f
    }                                            95cf
};                                               329b
```

5.6 Maximum matching of general graph (Edmond's blossom)

Usage:

`init(n)`	Initialize the template with n vertices, numbered from 1.
`add_edge(u, v)`	Add an undirected edge uv.
`solve()`	Find the maximum matching. Return the number of matched edges.
`mate[]`	The mate of a matched vertex. If it is not matched, then the value is 0.

Time Complexity: $O(|V|^3)$, but extremely fast in practice.

```
const int MAXN = 1024;
struct Blossom {
    vector<int> adj[MAXN];
    queue<int> q;
    int n;
    int label[MAXN], mate[MAXN], save[MAXN], used[MAXN];

    void init(int nv) {
        n = nv; for (auto& v : adj) v.clear();
        fill(range(label), 0); fill(range(mate), 0);
        fill(range(save), 0); fill(range(used), 0);
    }

    void add_edge(int u, int v) { adj[u].push_back(v); adj[v].push_back(u); }

    void rematch(int x, int y) {
        int m = mate[x]; mate[x] = y;
        if (mate[m] == x) {
            if (label[x] <= n) {
                mate[m] = label[x]; rematch(label[x], m);
            } else {
                int a = 1 + (label[x] - n - 1) / n;
                int b = 1 + (label[x] - n - 1) % n;
                rematch(a, b); rematch(b, a);
            }
        }
    }

    void traverse(int x) {
        Rep (i, n) save[i] = mate[i];
        rematch(x, x);
        Rep (i, n) {
            if (mate[i] != save[i]) used[i] ++;
            mate[i] = save[i];
        }
    }

    void relabel(int x, int y) {
        Rep (i, n) used[i] = 0;
        traverse(x); traverse(y);
        Rep (i, n) {
            if (used[i] == 1 and label[i] < 0) {
                label[i] = n + x + (y - 1) * n;
                q.push(i);
            }
        }
    }

    int solve() {
        Rep (i, n) {
            if (mate[i]) continue;
            Rep (j, n) label[j] = -1;
            label[i] = 0; q = queue<int>(); q.push(i);
            while (q.size()) {
                int x = q.front(); q.pop();
                for (int y : adj[x]) {
                    if (mate[y] == 0 and i != y) {
                        mate[y] = x; rematch(x, y); q = queue<int>(); break;
                    }
                    if (label[y] >= 0) { relabel(x, y); continue; }
                    if (label[mate[y]] < 0) {
                        label[mate[y]] = x; q.push(mate[y]);
                    }
                }
            }
        }
        int cnt = 0;
        Rep (i, n) cnt += (mate[i] > i);
        return cnt;
    }
};
```

5.7 Minimum cost maximum flow

```
struct edge{
    int from, to;
    int cap, flow;
    LL cost;
};

const LL INF = LLONG_MAX / 2;
const int MAXN = 5005;
struct MCMF {
    int s, t, n, m;
    vector<edge> edges;
```

```
b891     vector<int> G[MAXN];
f74f     bool inq[MAXN]; // queue
8f67     LL d[MAXN];     // distance
9524     int p[MAXN];    // previous
b330     int a[MAXN];    // improvement
427e
f7f2     void add_edge(int from, int to, int cap, LL cost) {
24f0         edges.push_back(edge{from, to, cap, 0, cost});
95f0         edges.push_back(edge{to, from, 0, 0, -cost});
fe77         m = edges.size();
dff5         G[from].push_back(m-2);
8f2d         G[to].push_back(m-1);
95cf     }
427e
3c52     bool spfa(){
93d2         queue<int> q;
8494         fill(d, d + MAXN, INF); d[s] = 0;
fd48         memset(inq, 0, sizeof(inq));
5e7c         q.push(s); inq[s] = true;
2dae         p[s] = 0; a[s] = INT_MAX;
cc78         while (!q.empty()){
b0aa             int u = q.front(); q.pop(); inq[u] = false;
3bba             for (int i : G[u]) {
56d8                 edge& e = edges[i];
3601                 if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
55bc                     d[e.to] = d[u] + e.cost;
ddf5                     p[e.to] = i;
8249                     a[e.to] = min(a[u], e.cap - e.flow);
e5d3                     if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
95cf                 }
95cf             }
95cf         }
6d7c         return d[t] != INF;
95cf     }
427e
71a4     void augment(){
06f1         int u = t;
b19d         while (u != s){
db09             edges[p[u]].flow += a[t];
25a9             edges[p[u]^1].flow -= a[t];
e6c9             u = edges[p[u]].from;
95cf         }
95cf     }
427e
```

```
#ifdef GIVEN_FLOW                                                  6e20
    bool min_cost(int s, int t, int f, LL& cost) {                 5972
        this->s = s; this->t = t;                                  590d
        int flow = 0;                                              21d4
        cost = 0;                                                  23cb
        while (spfa()) {                                           22dc
            augment();                                             bcdb
            if (flow + a[t] >= f){                                 a671
                cost += (f - flow) * d[t]; flow = f;               b14d
                return true;                                       3361
            } else {                                               8e2e
                flow += a[t]; cost += a[t] * d[t];                 2a83
            }                                                      95cf
        }                                                          95cf
        return false;                                              438e
    }                                                              95cf
#else                                                              a8cb
    int min_cost(int s, int t, LL& cost) {                         f9a9
        this->s = s; this->t = t;                                  590d
        int flow = 0;                                              21d4
        cost = 0;                                                  23cb
        while (spfa()) {                                           22dc
            augment();                                             bcdb
            flow += a[t]; cost += a[t] * d[t];                     2a83
        }                                                          95cf
        return flow;                                               84fb
    }                                                              95cf
#endif                                                             1937
};                                                                 329b
```

5.8 Fast LCA

All indices of the tree are 1-based.

Usage:

`preprocess(root)`	Initialize with tree rooted at `root`.
`lca(u, v)`	Query the lowest common ancestor of u and v.

```
const int MAXN = 500005;                                           0e34
vector<int> adj[MAXN];                                             0b32
int id[MAXN], nid;                                                 fccb
pair<int, int> st[MAXN << 1][33 - __builtin_clz(MAXN)];            1356
                                                                   427e
void dfs(int u, int p, int d) {                                    e16d
```

```
    st[id[u] = nid++][0] = {d, u};
    for (int v : adj[u]) {
        if (v == p) continue;
        dfs(v, u, d + 1);
        st[nid++][0] = {d, u};
    }
}

void preprocess(int root) {
    nid = 0;
    dfs(root, 0, 1);
    int l = 31 - __builtin_clz(nid);
    rep (j, l) rep (i, 1+nid-(1<<j))
        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
}

int lca(int u, int v) {
    tie(u, v) = minmax(id[u], id[v]);
    int k = 31 - __builtin_clz(v-u+1);
    return min(st[u][k], st[v-(1<<k)+1][k]).second;
}
```

5.9 Heavy-light decomposition

Time Complexity: The decomposition itself takes linear time. Each query takes $O(\log n)$ operations.

```
const int MAXN = 100005;
vector<int> adj[MAXN];
int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];

void dfs1(int x, int dep, int par){
    depth[x] = dep;
    sz[x] = 1;
    fa[x] = par;
    int maxn = 0, s = 0;
    for (int c: adj[x]){
        if (c == par) continue;
        dfs1(c, dep + 1, x);
        sz[x] += sz[c];
        if (sz[c] > maxn){
            maxn = sz[c];
            s = c;
        }
    }
    son[x] = s;
}

int cid = 0;
void dfs2(int x, int t){
    top[x] = t;
    id[x] = ++cid;
    if (son[x]) dfs2(son[x], t);
    for (int c: adj[x]){
        if (c == fa[x]) continue;
        if (c == son[x]) continue;
        else dfs2(c, c);
    }
}

void decomp(int root){
    dfs1(root, 1, 0);
    dfs2(root, root);
}

void query(int u, int v){
    while (top[u] != top[v]){
        if (depth[top[u]] < depth[top[v]]) swap(u, v);
        // id[top[u]] to id[u]
        u = fa[top[u]];
    }
    if (depth[u] > depth[v]) swap(u, v);
    // id[u] to id[v]
}
```

5.10 Centroid decomposition

Note that the centroid here is not the exact centroid of the graph. It only guarantees that the size of each subtree does not exceed half of that of the original tree. This is enough to guarantee the correct time complexity. All vertices are numbered from 1. Call `decomp(root)` to use.

Usage:

`decomp(u, p)`	Decompose the tree rooted at u with parent p.

Time Complexity: The decomposition itself takes $O(n \log n)$ time.

```
1fb6 vector<int> adj[100005];
88e0 int sz[100005], sum;
427e
f93d void getsz(int u, int p) {
5b36   sz[u] = 1; sum++;
18f6   for (int v : adj[u]) {
bd87     if (v == p) continue;
e3cb     getsz(v, u);
8449     sz[u] += sz[v];
95cf   }
95cf }
427e
67f9 int getcent(int u, int p) {
d51f   for (int v : adj[u])
76e4     if (v != p and sz[v] > sum / 2)
18e3       return getcent(v, u);
81b0   return u;
95cf }
427e
4662 void decompose(int u) {
618e   sum = 0; getsz(u, 0);
303c   u = getcent(u, 0); // update u to the centroid
427e
18f6   for (int v : adj[u]) {
427e     // get answer for subtree v
95cf   }
427e   // get answer for the whole tree
427e   // don't forget to count the centroid itself
427e
18f6   for (int v : adj[u]) { // divide and conquer
c375     adj[v].erase(find(range(adj[v]), u));
fa6b     decompose(v);
a717     adj[v].push_back(u); // restore deleted edge
95cf   }
95cf }
```

5.11 DSU on tree

This implementation avoids parallel existence of multiple data structures but requires that the data structure is invertible. To use this template, implement `merge`, `enter`, `leave` as needed; first call `decomp(root, 0)`, then call `work(root, 0, false)`. Labels of vertices start from 1.

Usage:

`decomp(u, p)`	Decompose the tree u.
`work(u, p, keep)`	Work for subtree u. When `keep` is set, information is not cleared.

Time Complexity: $O(n \log n)$ times the complexity for `merge`, `enter`, `leave`.

```
vector<int> adj[100005];                                         1fb6
int sz[100005], son[100005];                                     901d
                                                                 427e
void decomp(int u, int p) {                                      5559
    sz[u] = 1;                                                   50c0
    for (int v : adj[u]) {                                       18f6
        if (v == p) continue;                                    bd87
        decomp(v, u);                                            a851
        sz[u] += sz[v];                                          8449
        if (sz[v] > sz[son[u]]) son[u] = v;                      d28c
    }                                                            95cf
}                                                                95cf
                                                                 427e
template <typename T>                                            b7ec
void trav(T fn, int u, int p) {                                  62f5
    fn(u);                                                       4412
    for (int v : adj[u]) if (v != p) trav(fn, v, u);             30b3
}                                                                95cf
                                                                 427e
#define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])  7467
void work(int u, int p, bool keep) {                             33ff
    for_light(v) work(v, u, 0); // process light children        72a2
                                                                 427e
    // process heavy child                                       427e
    // current data structure contains info of heavy child       427e
    if (son[u]) work(son[u], u, 1);                              9866
                                                                 427e
    auto merge = [u] (int c) { /* count contribution of c */ };  18a9
    auto enter = [] (int c) { /* add vertex c */ };              1ab0
    auto leave = [] (int c) { /* remove vertex c*/ };            f241
                                                                 427e
    for_light(v) {                                               3d3b
        trav(merge, v, u);                                       74c6
        trav(enter, v, u);                                       c13d
    }                                                            95cf
                                                                 427e
    // count answer for root and add it                          427e
    // Warning: special check may apply to root!                 427e
```

```
c54f     merge(u);
9dec     enter(u);
427e 
427e     // leave current tree
4e3e     if (!keep) trav(leave, u, p);
95cf }
```

6 Data Structures

6.1 Fenwick tree (point update range query)

```
9976 struct bit_purq { // point update, range query
d7af     int N;
99ff     vector<LL> tr;
427e 
2d99     void init(int n) { tr.assign(N = n + 5, 0); }
427e 
63d0     LL sum(int n) {
f7ff         LL ans = 0;
6770         while (n) { ans += tr[n]; n &= n - 1; }
4206         return ans;
95cf     }
427e 
f4bd     void add(int n, LL x){
968e         while (n < N) { tr[n] += x; n += n & -n; }
95cf     }
329b };
```

6.2 Fenwick tree (range update point query)

```
3d03 struct bit_rupq{ // range update, point query
d7af     int N;
99ff     vector<LL> tr;
427e 
2d99     void init(int n) { tr.assign(N = n + 5, 0);}
427e 
38d4     LL query(int n) {
f7ff         LL ans = 0;
3667         while (n < N) { ans += tr[n]; n += n & -n; }
4206         return ans;
```

```
    }                                                    95cf
                                                         427e
    void add(int n, LL x) {                              f4bd
        while (n) { tr[n] += x; n &= n - 1; }            0a2b
    }                                                    95cf
};                                                       329b
```

6.3 Segment tree

```
LL p;                                                    3942
const int MAXN = 4 * 100006;                             1ebb
struct segtree {                                         451a
  int l[MAXN], m[MAXN], r[MAXN];                         27be
  LL val[MAXN], tadd[MAXN], tmul[MAXN];                  4510
                                                         427e
#define lson (o<<1)                                      ac35
#define rson (o<<1|1)                                    1294
                                                         427e
  void pull(int o) {                                     1344
    val[o] = (val[lson] + val[rson]) % p;                bbe9
  }                                                      95cf
                                                         427e
  void push_add(int o, LL x) {                           e4bc
    val[o] = (val[o] + x * (r[o] - l[o])) % p;           5dd6
    tadd[o] = (tadd[o] + x) % p;                         6eff
  }                                                      95cf
                                                         427e
  void push_mul(int o, LL x) {                           d658
    val[o] = val[o] * x % p;                             b82c
    tadd[o] = tadd[o] * x % p;                           aa86
    tmul[o] = tmul[o] * x % p;                           649f
  }                                                      95cf
                                                         427e
  void push(int o) {                                     b149
    if (l[o] == m[o]) return;                            3159
    if (tmul[o] != 1) {                                  0a90
      push_mul(lson, tmul[o]);                           0f4a
      push_mul(rson, tmul[o]);                           045e
      tmul[o] = 1;                                       ac0a
    }                                                    95cf
    if (tadd[o]) {                                       1b82
      push_add(lson, tadd[o]);                           9547
```

```
0e73       push_add(rson, tadd[o]);
6234       tadd[o] = 0;
95cf     }
95cf   }
427e
471c   void build(int o, int ll, int rr) {
0e87     int mm = (ll + rr) / 2;
9d27     l[o] = ll; r[o] = rr; m[o] = mm;
ac0a     tmul[o] = 1;
5c92     if (ll == mm) {
001f       scanf("%lld", val + o);
e5b6       val[o] %= p;
8e2e     } else {
7293       build(lson, ll, mm);
5e67       build(rson, mm, rr);
ba26       pull(o);
95cf     }
95cf   }
427e
4406   void add(int o, int ll, int rr, LL x) {
3c16     if (ll <= l[o] && r[o] <= rr) {
db32       push_add(o, x);
8e2e     } else {
c4b0       push(o);
4305       if (m[o] > ll) add(lson, ll, rr, x);
d5a6       if (m[o] < rr) add(rson, ll, rr, x);
ba26       pull(o);
95cf     }
95cf   }
427e
48cd   void mul(int o, int ll, int rr, LL x) {
3c16     if (ll <= l[o] && r[o] <= rr) {
e7d0       push_mul(o, x);
8e2e     } else {
c4b0       push(o);
d1ba       if (ll < m[o]) mul(lson, ll, rr, x);
67f3       if (m[o] < rr) mul(rson, ll, rr, x);
ba26       pull(o);
95cf     }
95cf   }
427e
0f62   LL query(int o, int ll, int rr) {
3c16     if (ll <= l[o] && r[o] <= rr) {
6dfe       return val[o];
```

```
    } else {                                                  8e2e
      push(o);                                                c4b0
      if (rr <= m[o]) return query(lson, ll, rr);             462a
      if (ll >= m[o]) return query(rson, ll, rr);             5cca
      return query(lson, ll, rr) + query(rson, ll, rr);       bbf9
    }                                                         95cf
  }                                                           95cf
} seg;                                                        4d99
```

6.4 Mo's algorithm

All intervals are closed on both sides. When running functions `enter()` and `leave()`, the global l and r has not changed yet. **Assume the data structure is initialized for empty interval**.

Usage:

`add_query(id, l, r)`	Add id-th query $[l, r]$.
`run()`	Run Mo's algorithm.
`yield(id)`	**TODO**. Yield answer for id-th query.
`enter(o)`	**TODO**. Add o-th element.
`leave(o)`	**TODO**. Remove o-th element.

```
constexpr int BLOCK_SZ = 300;                                      5194
                                                                   427e
struct query { int l, r, id; };                                    3ec4
vector<query> queries;                                             d26a
                                                                   427e
void add_query(int id, int l, int r) {                             1e30
  queries.push_back(query{l, r, id});                              54c9
}                                                                  95cf
                                                                   427e
int l, r;                                                          9f6b
                                                                   427e
// ----- functions to implement -----                              427e
inline void yield(int id);                                         50e1
inline void enter(int o);                                          b20d
inline void leave(int o);                                          13af
                                                                   427e
void run() {                                                       37f0
    if (queries.empty()) return;                                   ab0b
    sort(range(queries), [](query lhs, query rhs) {                8508
        int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;          c7f8
        if (lb != rb) return lb < rb;                              03e7
        return lhs.r < rhs.r;                                      0780
```

```cpp
    });
    l = queries[0].l;
    r = queries[0].r;
    for (int i = l; i <= r; i++) enter(i);
    for (query q : queries) {
        while (l > q.l) enter(--l);
        while (r < q.r) enter(++r);
        while (l < q.l) leave(l++);
        while (r > q.r) leave(r--);
        yield(q.id);
    }
}
```

6.5 Mo's algorithm on tree

Numbers of vertices are 1-based. Implement `deal(int u)` and `query::yield()`.

```cpp
const int MAXN = 200005, BLOCK = 300;
int n, m;
vector<int> adj[MAXN];
int en[MAXN], edx;
int dep[MAXN], fa[MAXN];
bool in[MAXN];

inline void deal(int u) {
    if (in[u] ^= 1) {
        // enter
    } else {
        // leave
    }
}

void moveto(int a, int b) {
    if (a == b) return;
    int cross = in[b] ? b : 0;
    auto moveup = [&] (int &x) {
        if (!cross) {
            if (in[x] and !in[fa[x]]) cross = x;
            else if (in[fa[x]] and !in[x]) cross = fa[x];
        }
        deal(x); x = fa[x];
    };
    while (dep[a] > dep[b]) moveup(a);
    while (dep[b] > dep[a]) moveup(b);
    while (a != b) moveup(a), moveup(b);
    deal(a); if (cross) deal(cross);
}

void dfs(int u, int p) {
    en[u] = edx++; fa[u] = p;
    for (int v : adj[u]) if (v != p) {
        dep[v] = dep[u] + 1;
        dfs(v, u); edx++;
    }
}

struct query {
    int l, r, id;
    void yield() { /* TODO */}
};
vector<query> qs;

void run() {
    dfs(1, 0);

    sort(range(qs), [] (query lhs, query rhs) {
        int u0 = en[lhs.l], v0 = en[rhs.l];
        int bl = u0 / BLOCK, br = v0 / BLOCK;
        if (bl != br) return bl < br;
        int u1 = en[lhs.r], v1 = en[rhs.r];
        return bl & 1 ? u1 < v1 : u1 > v1;
    });

    int l = 1, r = 1; deal(1);
    for (auto& q : qs) {
        moveto(l, q.l); l = q.l;
        moveto(r, q.r); r = q.r;
        q.yield();
    }
}
```

6.6 Treap

Self-balanced binary search tree which supports split and merge.
Usage:

`push(x)`	Push lazy tags to children.
`pull(x)`	Update statistics of node x.
`Init(x, v)`	Initialize node x with value v.
`Add(x, v)`	Apply addition to subtree x.
`Reverse(x)`	Apply reversion to subtree x.
`Merge(x, y)`	Merge trees rooted at x and y. Return the root of new tree.
`Split(t, k, x, y)`	Split out the left k elements of tree t. The roots of left part and right part are stored in x an d y, respectively.
`init(n)`	Initialize the treap with array of size n.
`work(op, l, r)`	Range operation over $[l, r)$.

Time Complexity: Expected $O(\log n)$ per operation.

```
const int MAXN = 200005;
mt19937 gen(time(NULL));
struct Treap {
    int ch[MAXN][2];
    int sz[MAXN], key[MAXN], val[MAXN];
    int add[MAXN], rev[MAXN];
    LL sum[MAXN] = {0};
    int maxv[MAXN] = {INT_MIN}, minv[MAXN] = {INT_MAX};

    void Init(int x, int v) {
        ch[x][0] = ch[x][1] = 0;
        key[x] = gen(); val[x] = v; pull(x);
    }

    void pull(int x) {
        sz[x] = 1 + sz[ch[x][0]] + sz[ch[x][1]];
        sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]];
        maxv[x] = max({val[x], maxv[ch[x][0]], maxv[ch[x][1]]});
        minv[x] = min({val[x], minv[ch[x][0]], minv[ch[x][1]]});
    }

    void Add(int x, int a) {
        val[x] += a; add[x] += a;
        sum[x] += LL(sz[x]) * a; maxv[x] += a; minv[x] += a;
    }

    void Reverse(int x) {
        rev[x] ^= 1;
        swap(ch[x][0], ch[x][1]);
    }

    void push(int x) {
        for (int c : ch[x]) if (c) {
            Add(c, add[x]);
            if (rev[x]) Reverse(c);
        }
        add[x] = 0; rev[x] = 0;
    }

    int Merge(int x, int y) {
        if (!x || !y) return x | y;
        push(x); push(y);
        if (key[x] > key[y]) {
            ch[x][1] = Merge(ch[x][1], y); pull(x); return x;
        } else {
            ch[y][0] = Merge(x, ch[y][0]); pull(y); return y;
        }
    }

    void Split(int t, int k, int &x, int &y) {
        if (t == 0) { x = y = 0; return; }
        push(t);
        if (sz[ch[t][0]] < k) {
            x = t; Split(ch[t][1], k - sz[ch[t][0]] - 1, ch[t][1], y);
        } else {
            y = t; Split(ch[t][0], k, x, ch[t][0]);
        }
        if (x) pull(x); if (y) pull(y);
    }
} treap;

int root;

void init(int n) {
    Rep (i, n) {
        int x; scanf("%d", &x);
        treap.Init(i, x);
        root = (i == 1) ? 1 : treap.Merge(root, i);
    }
}

void work(int op, int l, int r) {
    int tl, tm, tr;
    treap.Split(root, l, tl, tm);
    treap.Split(tm, r - l, tm, tr);
```

```
3658     if (op == 1) {
c039         int x; scanf("%d", &x); treap.Add(tm, x);
1dcb     } else if (op == 2) {
ae78         treap.Reverse(tm);
581d     } else if (op == 3) {
e092         printf("%lld␣%d␣%d\n",
867f                 treap.sum[tm], treap.minv[tm], treap.maxv[tm]);
95cf     }
6188     root = treap.Merge(treap.Merge(tl, tm), tr);
95cf }
```

6.7 Link/cut tree

Dynamic connectivity of undirected acyclic graph. Support single-vertex update, path aggregation and relative LCA query. Vertices are numbered from 1. Zero initialization is enough except for the statistic information.

Usage:

`pull(x)`	Update statistics of node x.
`Root(u)`	Get the root of tree where vertex u is in.
`Link(u, v)`	Link two unconnected trees.
`Cut(u, v)`	Cut an existent edge.
`Query(u, v)`	Path aggregation.
`Update(u, x)`	Single point modification.
`LCA(u, v, root)`	Get the lowest common ancestor of u and v in tree rooted at `root`.

Time Complexity: $O(\log n)$ per operation

```
2e73 const int MAXN = 1000005;
ca06 struct LCT {
6a6d     int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
c6e1     bool rev[MAXN];
427e
eba3     bool isroot(int x) { return ch[fa[x]][0] == x || ch[fa[x]][1] == x; }
f19f     void pull(int x) { sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]]; }
1c4d     void reverse(int x) { swap(ch[x][0], ch[x][1]); rev[x] ^= 1; }
1a53     void push(int x) {
89a0         if (rev[x]) rep (i, 2) if (ch[x][i]) reverse(ch[x][i]); rev[x] = 0;
95cf     }
425f     void rotate(int x) {
51af         int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
e1fe         if (isroot(y)) ch[z][ch[z][1] == y] = x;
1e6f         ch[x][!k] = y; ch[y][k] = w; if (w) fa[w] = y;
        fa[y] = x; fa[x] = z; pull(y);    6d09
    }    95cf
    void pushall(int x) { if (isroot(x)) pushall(fa[x]); push(x); }    52c6
    void splay(int x) {    f69c
        int y = x, z = 0;    d095
        for (pushall(y); isroot(x); rotate(x)) {    c494
            y = fa[x]; z = fa[y];    ceef
            if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);    4449
        }    95cf
        pull(x);    78a0
    }    95cf
    void access(int x) {    6229
        int z = x;    1548
        for (int y = 0; x; x = fa[y = x]) { splay(x); ch[x][1] = y; pull(x); }    8854
        splay(z);    7afd
    }    95cf
    void chroot(int x) { access(x); reverse(x); }    a067
    void split(int x, int y) { chroot(x); access(y); }    126d
    427e
    int Root(int x) {    d87a
        for (access(x); ch[x][0]; x = ch[x][0]) push(x);    f4f1
        splay(x); return x;    0d77
    }    95cf
    void Link(int u, int v) { chroot(u); fa[u] = v; }    9e46
    void Cut(int u, int v) { split(u, v); fa[u] = ch[v][0] = 0; pull(v); }    7c10
    int Query(int u, int v) { split(u, v); return sum[v]; }    0691
    void Update(int u, int x) { splay(u); val[u] = x; }    a999
    int LCA(int x, int y, int root) {    1f42
        chroot(root); access(x); splay(y);    6cb2
        while (fa[y]) splay(y = fa[y]);    02e5
        return y;    c218
    }    95cf
};    329b
```

6.8 Balanced binary search tree from `pb_ds`

```
#include <ext/pb_ds/assoc_container.hpp>    0475
using namespace __gnu_pbds;    332d
    427e
tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>    43a7
  rkt;
// null_tree_node_update    427e
```

```
427e
427e // SAMPLE USAGE
190e rkt.insert(x);          // insert element
05d4 rkt.erase(x);           // erase element
add5 rkt.order_of_key(x);    // obtain the number of elements less than x
b064 rkt.find_by_order(i);   // iterator to i-th (numbered from 0) smallest element
c103 rkt.lower_bound(x);
4ff4 rkt.upper_bound(x);
b19b rkt.join(rkt2);         // merge tree (only if their ranges do not intersect)
cb47 rkt.split(x, rkt2);     // split all elements greater than x to rkt2
```

6.9 Persistent segment tree, range k-th query

```
f1a7 struct node {
2ff6   static int n, pos;
427e
7cec   int value;
70e2   node *left, *right;
427e
20b0   void* operator new(size_t size);
427e
3dc0   static node* Build(int l, int r) {
b6c5     node* a = new node;
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
3ba2       a->left = Build(l, mid);
8aaf       a->right = Build(mid, r);
8e2e     } else {
bfc4       a->value = 0;
95cf     }
5ffd     return a;
95cf   }
427e
5a45   static node* init(int size) {
2c46     n = size;
7ee3     pos = 0;
be52     return Build(0, n);
95cf   }
427e
93c0   static int Query(node* lt, node *rt, int l, int r, int k) {
d30c     if (r == l + 1) return l;
181e     int mid = (l + r) / 2;
    if (rt->left->value - lt->left->value < k) {              cb5a
      k -= rt->left->value - lt->left->value;                 8edb
      return Query(lt->right, rt->right, mid, r, k);          2412
    } else {                                                  8e2e
      return Query(lt->left, rt->left, l, mid, k);            0119
    }                                                         95cf
  }                                                           95cf
                                                              427e
  static int query(node* lt, node *rt, int k) {               c9ad
    return Query(lt, rt, 0, n, k);                            9e27
  }                                                           95cf
                                                              427e
  node *Inc(int l, int r, int pos) const {                    b19c
    node* a = new node(*this);                                5794
    if (r > l + 1) {                                          ce96
      int mid = (l + r) / 2;                                  181e
      if (pos < mid)                                          203d
        a->left = left->Inc(l, mid, pos);                     f44a
      else                                                    649a
        a->right = right->Inc(mid, r, pos);                   1024
    }                                                         95cf
    a->value++;                                               2b3e
    return a;                                                 5ffd
  }                                                           95cf
                                                              427e
  node *inc(int index) {                                      e80f
    return Inc(0, n, index);                                  c246
  }                                                           95cf
} nodes[8000000];                                             865a
                                                              427e
int node::n, node::pos;                                       99ce
inline void* node::operator new(size_t size) {                1987
  return nodes + (pos++);                                     bb3c
}                                                             95cf
```

6.10 Block list

All indices are 0-based. All ranges are left-closed right-open.

Usage:

`block::fix()`	Apply tags to the current block.
`Init(l, r)`	Range initializer.
`Reverse(l, r)`	Reverse the range.
`Add(l, r, x)`	Add x to the range.
`Query(l, r)`	Range aggregation.

```
fd9e const int BLOCK = 800;
76b3 typedef vector<int> vi;
427e
a771 struct block {
8fbc     vi data;
e3b5     LL sum; int minv, maxv;
41db     int add; bool rev;
427e
d7eb     block(vi&& vec) : data(move(vec)),
1f0c         sum(accumulate(range(data), 0ll)),
8216         minv(*min_element(range(data))),
527d         maxv(*max_element(range(data))),
6437         add(0), rev(0) { }
427e
b919     void fix() {
0694         if (rev) reverse(range(data));          rev = 0;
0527         if (add) for (int& x : data) x += add;  add = 0;
95cf     }
427e
8bc4     void merge(block& another) {
b895         fix(); another.fix();
f516         vi temp(move(data));
d02c         temp.insert(temp.end(), range(another.data));
88ea         *this = block(move(temp));
95cf     }
427e
42e8     block split(int pos) {
3e79         fix();
ccab         block result(vi(data.begin() + pos, data.end()));
861a         data.resize(pos); *this = block(move(data));
56b0         return result;
95cf     }
329b };
427e
2a18 typedef list<block>::iterator lit;
427e
ce14 struct blocklist {
5540     list<block> blk;
```

```
                                                                            427e
    void maintain() {                                                       7b8e
        lit it = blk.begin();                                               3131
        while (it != blk.end() && next(it) != blk.end()) {                  4628
            lit it2 = it;                                                   852d
            while (next(it2) != blk.end() &&                                188c
                    it2->data.size() + next(it2)->data.size() <= BLOCK) {   3600
                it2->merge(*next(it2));                                     93e1
                blk.erase(next(it2));                                       e1fa
            }                                                               95cf
            ++it;                                                           5771
        }                                                                   95cf
    }                                                                       95cf
                                                                            427e
    lit split(int pos) {                                                    b7b3
        for (lit it = blk.begin(); ; it++) {                                2273
            if (pos == 0) return it;                                        5502
            while (it->data.size() > pos)                                   8e85
                blk.insert(next(it), it->split(pos));                       2099
            pos -= it->data.size();                                         a5a1
                                                                            427e
        }                                                                   95cf
    }                                                                       95cf
                                                                            427e
    void Init(int *l, int *r) {                                             1c7b
        for (int *cur = l; cur < r; cur += BLOCK)                           9919
            blk.emplace_back(vi(cur, min(cur + BLOCK, r)));                 8950
    }                                                                       95cf
                                                                            427e
    void Reverse(int l, int r) {                                            a22f
        lit it = split(l), it2 = split(r);                                  997b
        reverse(it, it2);                                                   dfd0
        while (it != it2) {                                                 8f89
            it->rev ^= 1;                                                   6a06
            it++;                                                           5283
        }                                                                   95cf
        maintain();                                                         b204
    }                                                                       95cf
                                                                            427e
    void Add(int l, int r, int x) {                                         3cce
        lit it = split(l), it2 = split(r);                                  997b
        while (it != it2) {                                                 8f89
            it->sum += LL(x) * it->data.size();                             e927
            it->minv += x; it->maxv += x;                                   03d3
```

```
4511             it->add += x; it++;
95cf         }
b204         maintain();
95cf     }
427e
3ad3     void Query(int l, int r) {
997b         lit it = split(l), it2 = split(r);
c33d         LL sum = 0; int minv = INT_MAX, maxv = INT_MIN;
8f89         while (it != it2) {
e472             sum += it->sum;
72c4             minv = min(minv, it->minv);
e1c4             maxv = max(maxv, it->maxv);
5283             it++;
95cf         }
b204         maintain();
8792         printf("%lld␣%d␣%d\n", sum, minv, maxv);
95cf     }
958e } lst;
```

6.11 Persistent block list

Block list that supports persistence. All indices are 0-based. All ranges are left-closed right-open. `std::shared_ptr` is used to ease memory management. One should modify the constructor of `block` to maintain extra information. Here we use this policy that the size of each block does not exceed `BLOCK`, while the sum of sizes of two adjacent blocks does not less than `BLOCK`.
When some operation that breaks block list property, please call `maintain` in time to restore the property.
Usage:

`maintain()`	Maintain the block list property.
`split(pos)`	Split the block list at position `pos`. Returns an iterator to a block starting at `pos`.
`sum(l, r)`	An example function of list traversal between $[l, r)$.

Time Complexity: When `BLOCK` is properly selected, the time complexity is $O(\sqrt{n})$ per operation.

```
a19e constexpr int BLOCK = 800;
76b3 typedef vector<int> vi;
0563 typedef shared_ptr<vi> pvi;
013b typedef shared_ptr<const vi> pcvi;
427e
a771 struct block {
    pcvi data;                                                              2989
    LL sum;                                                                 8fd0
                                                                            427e
    // add information to maintain                                          427e
    block(pcvi ptr) :                                                       a613
        data(ptr),                                                          24b5
        sum(accumulate(ptr->begin(), ptr->end(), 0ll))                      0cf0
    { }                                                                     e93b
                                                                            427e
    void merge(const block& another) {                                      5c0f
        pvi temp = make_shared<vi>(data->begin(), data->end());             0b18
        temp->insert(temp->end(), another.data->begin(), another.data->end()); ac21
        *this = block(temp);                                                6467
    }                                                                       95cf
                                                                            427e
    block split(int pos) {                                                  42e8
        block result(make_shared<vi>(data->begin() + pos, data->end()));    dac1
        *this = block(make_shared<vi>(data->begin(), data->begin() + pos)); 01db
        return result;                                                      56b0
    }                                                                       95cf
};                                                                          329b
                                                                            427e
typedef list<block>::iterator lit;                                          2a18
                                                                            427e
struct blocklist {                                                          ce14
    list<block> blk;                                                        5540
                                                                            427e
    void maintain() {                                                       7b8e
        lit it = blk.begin();                                               3131
        while (it != blk.end() and next(it) != blk.end()) {                 5e44
            lit it2 = it;                                                   852d
            while (next(it2) != blk.end() and                               0b03
                    it2->data->size() + next(it2)->data->size() <= BLOCK) { 029f
                it2->merge(*next(it2));                                     93e1
                blk.erase(next(it2));                                       e1fa
            }                                                               95cf
            ++it;                                                           5771
        }                                                                   95cf
    }                                                                       95cf
                                                                            427e
    lit split(int pos) {                                                    b7b3
        for (lit it = blk.begin(); ; it++) {                                2273
            if (pos == 0) return it;                                        5502
            while (it->data->size() > pos) {                                d480
```

```
                blk.insert(next(it), it->split(pos));
            }
            pos -= it->data->size();
        }
    }

    LL sum(int l, int r) { // traverse
        lit it1 = split(l), it2 = split(r);
        LL res = 0;
        while (it1 != it2) {
            res += it1->sum;
            it1++;
        }
        maintain();
        return res;
    }
};
```

6.12 Sparse table, range minimum query

The array is 0-based and the range is left-closed right-open.

```
const int MAXN = 100007;
int a[MAXN], st[MAXN][30];

void init(int n){
    int l = log2(n);
    rep (i, n) st[i][0] = a[i];
    rep (j, l) rep (i, 1+n-(1<<j))
        st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
}

int rmq(int l, int r){
    int k = log2(r - l);
    return min(st[l][k], st[r-(1<<k)][k]);
}
```

7 Geometrics

7.1 2D geometric template

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

typedef int T;
typedef struct pt {
    T x, y;
    T operator , (pt a) { return x*a.x + y*a.y; } // inner product
    T operator * (pt a) { return x*a.y - y*a.x; } // outer product
    pt operator + (pt a) { return {x+a.x, y+a.y}; }
    pt operator - (pt a) { return {x-a.x, y-a.y}; }

    pt operator * (T k) { return {x*k, y*k}; }
    pt operator - () { return {-x, -y};}
} vec;

typedef pair<pt, pt> seg;

bool ptOnSeg(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    return (v1, v2) <= 0 && v1 * v2 == 0;
}

// 0 not on segment
// 1 on segment except vertices
// 2 on vertices
int ptOnSeg2(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    T ip = (v1, v2);
    if (v1 * v2 != 0 || ip > 0) return 0;
    return (v1, v2) ? 1 : 2;
}

// if two orthogonal rectangles do not touch, return true
inline bool nIntRectRect(seg a, seg b){
    return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||
           min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||
           min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||
           min(b.first.y, b.second.y) > max(a.first.y, a.second.y);
}

// >0 in order
// <0 out of order
// =0 not standard
```

```
7538 inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}
427e 
31ed inline bool intersect(seg a, seg b){
427e     // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
           and b are non-collinear
cb52     return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
059e            rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
95cf }
427e 
427e // 0 not insersect
427e // 1 standard intersection
427e // 2 vertex-line intersection
427e // 3 vertex-vertex intersection
427e // 4 collinear and have common point(s)
4d19 int intersect2(seg& a, seg& b){
5dc4     if (nIntRectRect(a, b)) return 0;
42c0     vec va = a.second - a.first, vb = b.second - b.first;
2096     double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
72fe            j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
5ac6     if (j1 < 0 || j2 < 0) return 0;
9400     if (j1 != 0 && j2 != 0) return 1;
83db     if (j1 == 0 && j2 == 0){
6b0c         if (va * vb == 0) return 4; else return 3;
fb17     } else return 2;
95cf }
427e 
2c68 template <typename Tp = T>
5894 inline pt getIntersection(pt P, vec v, pt Q, vec w){
6850     static_assert(is_same<Tp, double>::value, "must␣be␣double!");
7c9a     return P + v * (w*(P-Q)/(v*w));
95cf }
427e 
427e // -1 outside the polygon
427e // 0  on the border of the polygon
427e // 1  inside the polygon
cbdd int ptOnPoly(pt p, pt* poly, int n){
5fb4     int wn = 0;
1294     for (int i = 0; i < n; i++) {
427e 
3cae         T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
b957         if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){
8c40             if (k > 0 && d1 <= 0 && d2 > 0) wn++;
3c4d             if (k < 0 && d2 <= 0 && d1 > 0) wn--;
aad3         } else return 0;
```

```
    }                                                    95cf
    return wn ? 1 : -1;                                  0a5f
}                                                        95cf
                                                         427e
istream& operator >> (istream& lhs, pt& rhs){            d4a3
    lhs >> rhs.x >> rhs.y;                               fa86
    return lhs;                                          331a
}                                                        95cf
                                                         427e
istream& operator >> (istream& lhs, seg& rhs){           07ae
    lhs >> rhs.first >> rhs.second;                      5cab
    return lhs;                                          331a
}                                                        95cf
```

8 Appendices

8.1 Primes

8.1.1 First primes

p	$g(p)$	p	$g(p)$	p	$g(p)$	p	$g(p)$	p	$g(p)$
2	1	3	2	5	2	7	3	11	2
13	2	17	3	19	2	23	5	29	2
31	3	37	2	41	6	43	3	47	5
53	2	59	2	61	2	67	2	71	7
73	5	79	3	83	2	89	3	97	5
101	2	103	5	107	2	109	6	113	3
127	3	131	2	137	3	139	2	149	2
151	6	157	5	163	2	167	5	173	2
179	2	181	2	191	19	193	5	197	2
199	3	211	2	223	3	227	2	229	6

8.1.2 Arbitrary length primes

$\lg p$	p	$g(p)$	p	$g(p)$
3	967	5	1031	14
4	9859	2	10273	10
5	96331	10	102931	3
6	958543	6	1031137	5
7	9594539	2	10169651	2
8	96243449	3	103211039	7
9	980483981	2	1042484357	2
10	9858935453	2	10261276009	7
11	95748666809	3	101759940101	2
12	950781833849	3	1012797784423	5
13	9739822952371	7	10037217092377	7
14	96181051140397	5	104974966380359	11
15	981030138360889	13	1029038416465403	2
16	9655206098080843	3	10116299875820773	2
17	97687777921994419	3	101506415998163437	2

8.1.3 $\sim 1 \times 10^9$

p	$g(p)$	p	$g(p)$	p	$g(p)$
954854573	3	967607731	2	973215833	3
975831713	3	978949117	2	980766497	3
983879921	3	985918807	3	986608921	29
991136977	5	991752599	13	997137961	11
1003911991	3	1009775293	2	1012423549	6
1021000537	5	1023976897	7	1024153643	2
1037027287	3	1038812881	11	1044754639	3
1045125617	3	1047411427	3	1047753349	6

8.1.4 $\sim 1 \times 10^{18}$

p	$g(p)$	p	$g(p)$
951970612352230049	3	963284339889659609	3
967495386904694119	3	969751761517096213	2
983238274281901499	2	984647442475101409	23
989286107138674069	11	1002507954383424641	3
1006658951440146419	2	1020152326159075903	3
1034876265966119449	7	1042753851435034019	2
1043609016597371563	2	1045571042176595707	2
1048364250160580293	2	1049495624119026949	2

8.2 Pell's equation

$x^2 - ny^2 = 1$, where n is a positive nonsquare integer.
Let (x_0, y_0) be the smallest positive solution of the equation, then the k-th solution is:

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_0 & ny_0 \\ y_0 & x_0 \end{pmatrix}^k \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Some smallest solutions to Pell's equation:

n	2	3	5	6	7	8	10	11	12	13	14	15	17	18	19	20
x	3	2	9	5	8	3	19	10	7	649	15	4	33	17	170	9
y	2	1	4	2	3	1	6	3	2	180	4	1	8	4	39	2

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

where G is a group acting on X, X^g is the set of elements in X that are fixed by g, i.e. $X^g = \{x \in X : gx = x\}$.

The unweighted version of Pólya enumeration theorem says that

$$|Y^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c_g}$$

where $m = |X|$ is the number of colors, c_g is the number of the cycles of permutation g.

8.4 Supnick TSP

Given f and $x_1 \le x_2 \le \cdots \le x_n$, if f is Supnick, then

$$\sum_{i=1}^{n} f(x_{\pi(i)}, x_{\pi(i+1)})$$

1. is minimized when $\pi = (1, 3, 5, 7, \cdots, 8, 6, 4, 2)$.
2. is maximized when $\pi = (n, 2, n-2, 4, \cdots, 5, n-3, 3, n-1, 1)$.

8.5 Lagrange's interpolation

For sample points $(x_0, y_0), \cdots, (x_k, y_k)$, define

$$l_j(x) = \prod_{0 \le m \le k, m \ne j} \frac{x - x_m}{x_j - x_m}$$

then the Lagrange polynomial is

$$L(x) = \sum_{j=0}^{k} y_j l_j(x).$$

To use the script below, type two lines

```
x0 x1 x2 ... xn
y0 y1 y2 ... yn
```

the script will print the fractional coefficient of the polynomial in ascending exponent order.

```
#!/usr/bin/python2                                               6dc9
from fractions import *                                          4b2b
                                                                 427e
def polymul(a, b) :                                              796b
    p = [0] * (len(a)+len(b)-1)                                  83e4
    for e1, c1 in enumerate(a) :                                 f697
        for e2, c2 in enumerate(b) :                             156c
            p[e1+e2] += c1*c2                                    dfce
    return p                                                     5849
                                                                 427e
x, y = [map(Fraction, raw_input().split()) for _ in 0,0]         f06d
n = len(x)                                                       e80a
lj = [reduce(polymul, [[-x[m]/(x[j]-x[m]), 1/(x[j]-x[m])]        a649
    for m in range(n) if m != j]) for j in range(n)]             9dfa
print '␣'.join(map(str, map(sum, zip(*map(                       3cae
    lambda a, b : [x*a for x in b], y, lj)))))                   7c0d
```