南京大学 ACM-ICPC 集训队代码模版库

OS: Linux-4.15.0-45-generic-x86_64-with-Ubuntu-18.04-bionic
python: 2.7.15rc1
2019-02-25 18:27:33.343420, build 0021

Contents

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

```
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;
1ce6         for (int i=0; i<G[x].size(); i++)
d942             if (!dfs(G[x][i])) 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);
6835         G[y^1].push_back(x);
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;
f4b9                 if (!dfs(i)){
3f03                     while (c > 0) mark[S[--c]] = false;
86c5                     if (!dfs(i+1)) return false;
95cf                 }
95cf             }
            }
            return true;
        }

        inline bool value(unsigned i){return mark[2*i+1];}
    };
```

```
    }                                                          95cf
    return true;                                               3361
}                                                              95cf
                                                               427e
inline bool value(unsigned i){return mark[2*i+1];}             5f0a
};                                                             329b
```

2.2 Knuth's optimization

```
int n;                                                         5c83
int dp[256][256], dc[256][256];                                d77c
                                                               427e
template <typename T>                                          b7ec
void compute(T cost) {                                         0bc7
  for (int i = 0; i <= n; i++) {                               0423
    dp[i][i] = 0;                                              8f5e
    dc[i][i] = i;                                              9488
  }                                                            95cf
  rep (i, n) {                                                 be8e
    dp[i][i+1] = 0;                                            95b5
    dc[i][i+1] = i;                                            aa0f
  }                                                            95cf
  for (int len = 2; len <= n; len++) {                         ec08
    for (int i = 0; i + len <= n; i++) {                       88b8
      int j = i + len;                                         d3da
      int lbnd = dc[i][j-1], rbnd = dc[i+1][j];                9824
      dp[i][j] = INT_MAX / 2;                                  a24a
      int c = cost(i, j);                                      f933
      for (int k = lbnd; k <= rbnd; k++) {                     90d2
        int res = dp[i][k] + dp[k][j] + c;                     9bd0
        if (res < dp[i][j]) {                                  26b5
          dp[i][j] = res;                                      e6af
          dc[i][j] = k;                                        9c88
        }                                                      95cf
      }                                                        95cf
    }                                                          95cf
  }                                                            95cf
};                                                             329b
```

2.3 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.

Usage:

`add_query(id, l, r)`	Add id-th query $[l, r]$.
`run()`	Run Mo's algorithm.
`init()`	**TODO**. Initialize the range $[l, r]$.
`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;

struct query { int l, r, id; };
vector<query> queries;

void add_query(int id, int l, int r) {
  queries.push_back(query{l, r, id});
}

int l, r;

// ----- functions to implement -----
inline void init();
inline void yield(int id);
inline void enter(int o);
inline void leave(int o);

void run() {
  if (queries.empty()) return;
  sort(range(queries), [](query lhs, query rhs) {
    int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;
    if (lb != rb) return lb < rb;
    return lhs.r < rhs.r;
  });
  l = queries[0].l;
  r = queries[0].r;
  init();
  for (query q : queries) {
    while (l > q.l) enter(l - 1), l--;
    while (r < q.r) enter(r + 1), r++;
    while (l < q.l) leave(l), l++;
    while (r > q.r) leave(r), r--;
    yield(q.id);
  }
}
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
const int SIZE = 10005;

struct kmp_matcher {
  char p[SIZE];
  int fail[SIZE];
  int len;

  void construct(const char* needle) {
    len = strlen(p);
    strcpy(p, needle);
    fail[0] = fail[1] = 0;
    for (int i = 1; i < len; i++) {
      int j = fail[i];
      while (j && p[i] != p[j]) j = fail[j];
      fail[i + 1] = p[i] == p[j] ? j + 1 : 0;
    }
  }

  inline void found(int pos) {
    // ! add codes for having found at pos
  }

  void match(const char* haystack) {  // must be called after construct
    const char* t = haystack;
    int n = strlen(t);
    int j = 0;
    rep(i, n) {
      while (j && p[j] != t[i]) j = fail[j];
      if (p[j] == t[i]) j++;
      if (j == len) found(i - len + 1);
    }
  }
};
```

3.2 Manacher algorithm

```cpp
struct Manacher {
  int Len;
  vector<int> lc;
  string s;

  void work() {
    lc[1] = 1;
    int k = 1;

    for (int i = 2; i <= Len; i++) {
      int p = k + lc[k] - 1;
      if (i <= p) {
        lc[i] = min(lc[2 * k - i], p - i + 1);
      } else {
        lc[i] = 1;
      }
      while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
      if (i + lc[i] > k + lc[k]) k = i;
    }
  }

  void init(const char *tt) {
    int len = strlen(tt);
    s.resize(len * 2 + 10);
    lc.resize(len * 2 + 10);
    s[0] = '*';
    s[1] = '#';
    for (int i = 0; i < len; i++) {
      s[i * 2 + 2] = tt[i];
      s[i * 2 + 1] = '#';
    }
    s[len * 2 + 1] = '#';
    s[len * 2 + 2] = '\0';
    Len = len * 2 + 2;
    work();
  }

  pair<int, int> maxpal(int l, int r) {
    int center = l + r + 1;
    int rad = lc[center] / 2;
    int rmid = (l + r + 1) / 2;
    int rl = rmid - rad, rr = rmid + rad - 1;
    if ((r ^ l) & 1) {
    } else rr++;
    return {max(l, rl), min(r, rr)};
  }
};
```

3.3 Aho-corasick automaton

```cpp
struct AC : Trie {
  int fail[MAXN];
  int last[MAXN];

  void construct() {
    queue<int> q;
    fail[0] = 0;
    rep(c, CHARN) {
      if (int u = tr[0][c]) {
        fail[u] = 0;
        q.push(u);
        last[u] = 0;
      }
    }
    while (!q.empty()) {
      int r = q.front();
      q.pop();
      rep(c, CHARN) {
        int u = tr[r][c];
        if (!u) {
          tr[r][c] = tr[fail[r]][c];
          continue;
        }
        q.push(u);
        int v = fail[r];
        while (v && !tr[v][c]) v = fail[v];
        fail[u] = tr[v][c];
        last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
      }
    }
  }

  void found(int pos, int j) {
```

```
043e     if (j) {
427e       // ! add codes for having found word with tag[j]
4a96       found(pos, last[j]);
95cf     }
95cf   }
427e
9785   void find(const char* text) {  // must be called after construct()
80a4     int p = 0, c, len = strlen(text);
9c94     rep(i, len) {
b3db       c = id(text[i]);
f119       p = tr[p][c];
f08e       if (tag[p])
389b         found(i, p);
1e67       else if (last[p])
299e         found(i, last[p]);
95cf     }
95cf   }
329b };
```

3.4 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

```
de09 void radix_sort(int x[], int y[], int sa[], int n, int m) {
ec00     static int cnt[1000005];    // size > max(n, m)
6066     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);
    iota(y, y + n, 0);                                                                afbb
    radix_sort(rk, y, sa, n, m);                                                      7b42
    for (int j = 1, p = 0; j <= n; j <<= 1, m = p, p = 0) {                           c8c2
        for (int i = n - j; i < n; i++) y[p++] = i;                                   8c3a
        rep (i, n) if (sa[i] >= j) y[p++] = sa[i] - j;                                9323
        radix_sort(rk, y, sa, n, m + 1);                                              9e9d
        swap_ranges(rk, rk + n, y);                                                   ae41
        rk[sa[0]] = p = 1;                                                            ffd2
        for (int i = 1; i < n; i++)                                                   445e
            rk[sa[i]] = ((y[sa[i]] == y[sa[i-1]] and y[sa[i]+j] == y[sa[i-1]+j])      f8dc
                ? p : ++p);
        if (p == n) break;                                                            02f0
    }                                                                                 95cf
    rep (i, n) rk[sa[i]] = i;                                                         97d9
}                                                                                     95cf
                                                                                      427e
void calc_height(int s[], int sa[], int rk[], int h[], int n) {                       1715
    int k = 0;                                                                        c41f
    h[0] = 0;                                                                         f313
    rep (i, n) {                                                                      be8e
        k = max(k - 1, 0);                                                            0883
        if (rk[i]) while (s[i+k] == s[sa[rk[i]-1]+k]) ++k;                            527d
        h[rk[i]] = k;                                                                 56b7
    }                                                                                 95cf
}                                                                                     95cf
```

3.5 Trie

```
const int MAXN = 12000;                                                               e6f1
const int CHARN = 26;                                                                 dd87
                                                                                      427e
inline int id(char c) { return c - 'a'; }                                             8ff5
                                                                                      427e
struct Trie {                                                                         a281
  int n;                                                                              5c83
  int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail                                  f4f5
  int tag[MAXN];                                                                      35a5
                                                                                      427e
  Trie() {                                                                            4fee
    memset(tr[0], 0, sizeof(tr[0]));                                                  3ccc
    tag[0] = 0;                                                                       4d52
    n = 1;                                                                            46bf
```

```
95cf    }
427e
427e    // tag should not be 0
30b0    void add(const char* s, int t) {
d50a      int p = 0, c, len = strlen(s);
9c94      rep(i, len) {
3140        c = id(s[i]);
d6c8        if (!tr[p][c]) {
26dd          memset(tr[n], 0, sizeof(tr[n]));
2e5c          tag[n] = 0;
73bb          tr[p][c] = n++;
95cf        }
f119        p = tr[p][c];
95cf      }
35ef      tag[p] = t;
95cf    }
427e
427e    // returns 0 if not found
427e    // AC automaton does not need this function
216c    int search(const char* s) {
d50a      int p = 0, c, len = strlen(s);
9c94      rep(i, len) {
3140        c = id(s[i]);
f339        if (!tr[p][c]) return 0;
f119        p = tr[p][c];
95cf      }
840e      return tag[p];
95cf    }
329b  };
```

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;
0291  LL pg[MAXN];
427e
dfe7  inline LL mul(LL x, LL y) { return __int128_t(x) * y % mod; }
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;             3cd3
  }                                             95cf
  return (q + ret) % q;                         2e47
}                                               95cf
```

4.2 Matrix powermod

```
44b4 const int MAXN = 105;
92df const LL modular = 1000000007;
5c83 int n; // order of matrices
427e
8864 struct matrix{
3180     LL m[MAXN][MAXN];
427e
43c5     void operator *=(matrix& a){
e735         static LL t[MAXN][MAXN];
34d7         Rep (i, n){
4c11             Rep (j, n){
ee1e                 t[i][j] = 0;
c4a7                 Rep (k, n){
fcaf                     t[i][j] += (m[i][k] * a.m[k][j]) % modular;
199e                     t[i][j] %= modular;
95cf                 }
95cf             }
95cf         }
dad4         memcpy(m, t, sizeof(t));
95cf     }
329b };
427e
63d8 matrix r;
3ec2 void m_powmod(matrix& b, LL e){
83f0     memset(r.m, 0, sizeof(r.m));
a7c3     Rep(i, n)
de64         r.m[i][i] = 1;
3e90     while (e){
5a0e         if (e & 1) r *= b;
35c5         b *= b;
16fc         e >>= 1;
95cf     }
95cf }
```

4.3 Linear basis

```
8b44 const int MAXD = 30;
03a6 struct linearbasis {
3558     ULL b[MAXD] = {};
427e
    bool insert(LL v) {                                        1566
        for (int j = MAXD - 1; j >= 0; j--) {                  9b2b
            if (!(v & (1ll << j))) continue;                   de36
            if (b[j]) v ^= b[j]                                ee78
            else {                                             037f
                for (int k = 0; k < j; k++)                    7836
                    if (v & (1ll << k)) v ^= b[k];             f0b4
                for (int k = j + 1; k < MAXD; k++)             b0aa
                    if (b[k] & (1ll << j)) b[k] ^= v;          46c9
                b[j] = v;                                      8295
                return true;                                   3361
            }                                                  95cf
        }                                                      95cf
        return false;                                          438e
    }                                                          95cf
};                                                             329b
```

4.4 Gauss elimination over finite field

```
const LL p = 1000000007;                                       b784
                                                               427e
LL powmod(LL b, LL e) {                                        2a2c
  LL r = 1;                                                    95a2
  while (e) {                                                  3e90
    if (e & 1) r = r * b % p;                                  1783
    b = b * b % p;                                             5549
    e >>= 1;                                                   16fc
  }                                                            95cf
  return r;                                                    547e
}                                                              95cf
                                                               427e
typedef vector<LL> VLL;                                        c130
typedef vector<VLL> VVLL;                                      42ac
                                                               427e
LL gauss(VVLL &a, VVLL &b) {                                   2c62
  const int n = a.size(), m = b[0].size();                     561b
  vector<int> irow(n), icol(n), ipiv(n);                       a25e
  LL det = 1;                                                  2976
                                                               427e
  rep (i, n) {                                                 be8e
    int pj = -1, pk = -1;                                      d2b5
    rep (j, n) if (!ipiv[j])                                   6b4a
```

```
e582       rep (k, n) if (!ipiv[k])
6112         if (pj == -1 || a[j][k] > a[pj][pk]) {
a905           pj = j;
657b           pk = k;
95cf         }
d480     if (a[pj][pk] == 0) return 0;
0305     ipiv[pk]++;
8dad     swap(a[pj], a[pk]);
aad8     swap(b[pj], b[pk]);
be4d     if (pj != pk) det = (p - det) % p;
d080     irow[i] = pj;
f156     icol[i] = pk;
427e
4ecd     LL c = powmod(a[pk][pk], p - 2);
865b     det = det * a[pk][pk] % p;
c36a     a[pk][pk] = 1;
dd36     rep (j, n) a[pk][j] = a[pk][j] * c % p;
1b23     rep (j, m) b[pk][j] = b[pk][j] * c % p;
f8f3     rep (j, n) if (j != pk) {
e97f       c = a[j][pk];
c449       a[j][pk] = 0;
820b       rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
f039       rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
95cf     }
95cf   }
427e
37e1   for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
50dc     for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
95cf   }
f27f   return det;
95cf }
```

4.5 Berlekamp-Massey algorithm

```
d790 vector<int> berlekamp(const vector<int>& a) {
4166     vector<int> p = {1}, r = {1};
baed     int dif = 1;
8bc9     rep (i, a.size()) {
3e58         int u = 0;
ac8e         rep (j, p.size())
a488             u = (u + 1ll * p[j] * a[i-j]) % mod;
eae9         if (u == 0) {
            r.insert(r.begin(), 0);                                          b14c
        } else {                                                             8e2e
            auto op = p;                                                     0c78
            p.resize(max(p.size(), r.size() + 1));                           02f6
            int idif = inv(dif);                                             786b
            rep (j, r.size())                                                9b57
                p[j+1] =                                                     793c
                    (p[j+1] - 1ll * r[j] * idif % mod * u % mod + mod) % mod;  1836
            dif = u;                                                         644c
            r = op;                                                          bc58
        }                                                                    95cf
    }                                                                        95cf
    return p;                                                                e149
}                                                                            95cf
```

4.6 Fast Walsh-Hadamard transform

```
void fwt(int* a, int n){                                                     061e
    for (int d = 1; d < n; d <<= 1)                                          5595
        for (int i = 0; i < n; i += d << 1)                                  05f2
            rep (j, d){                                                      b833
                int x = a[i+j], y = a[i+j+d];                                7796
                // 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                    427e
            }                                                                95cf
}                                                                            95cf
                                                                             427e
void ifwt(int* a, int n){                                                    4db1
    for (int d = 1; d < n; d <<= 1)                                          5595
        for (int i = 0; i < n; i += d << 1)                                  05f2
            rep (j, d){                                                      b833
                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
```

```
8a42     rep(i, n) a[i] *= b[i];
430f     ifwt(a, n);
95cf }
```

4.7 Fast fourier transform

```
4e09 const int NMAX = 1<<20;
427e
3fbf typedef complex<double> cplx;
427e
abd1 const double PI = 2*acos(0.0);
12af struct FFT{
c47c     int rev[NMAX];
27d7     cplx omega[NMAX], oinv[NMAX];
9827     int K, N;
427e
1442     FFT(int k){
e209         K = k; N = 1 << k;
b393         rep (i, N){
7ba3             rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
1908             omega[i] = polar(1.0, 2.0 * PI / N * i);
a166             oinv[i] = conj(omega[i]);
95cf         }
95cf     }
427e
b941     void dft(cplx* a, cplx* w){
a215         rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e         for (int l = 2; l <= N; l *= 2){
2969             int m = l/2;
b3cf             for (cplx* p = a; p != a + N; p += l)
c24f                 rep (k, m){
fe06                     cplx t = w[N/l*k] * p[k+m];
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
    void conv(cplx* a, cplx* b){                         bdc0
        fft(a); fft(b);                                  6497
        rep (i, N) a[i] *= b[i];                         12a5
        ifft(a);                                         f84e
    }                                                    95cf
};                                                       329b
```

4.8 Number theoretic transform

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

4.9 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;
```

```
            if (i % prime[j] == 0) break;    5f51
        }                                    95cf
    }                                        95cf
}                                            95cf
```

4.10 Sieve of Euler (General)

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

4.11 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 $n < 2^{64}$.

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

4.12 Pollard's rho algorithm

```
2e6b  ULL gcd(ULL a, ULL b) {return b ? gcd(b, a % b) : a;}
427e
54a5  ULL PollardRho(ULL n){
45eb      ULL c, x, y, d = n;
```

```
    if (~n&1) return 2;                          d3e5
    while (d == n){                              3c69
        x = y = 2;                               0964
        d = 1;                                   4753
        c = rand() % (n - 1) + 1;                5952
        while (d == 1){                          9e5b
            x = (mulmod(x, x, n) + c) % n;       33d5
            y = (mulmod(y, y, n) + c) % n;       e1bf
            y = (mulmod(y, y, n) + c) % n;       e1bf
            d = gcd(x>y ? x-y : y-x, n);         a313
        }                                        95cf
    }                                            95cf
    return d;                                    5d89
}                                                95cf
```

5 Graph Theory

5.1 Strongly connected component

```
const int MAXV = 100005;                         837c
                                                 427e
struct graph{                                    2ea0
    vector<int> adj[MAXV];                       88e3
    stack<int> s;                                9cad
    int V; // number of vertices                 3d02
    int pre[MAXV], lnk[MAXV], scc[MAXV];         8b6c
    int time, sccn;                              27ee
                                                 427e
    void add_edge(int u, int v){                 bfab
        adj[u].push_back(v);                     c71a
    }                                            95cf
                                                 427e
    void dfs(int u){                             d714
        pre[u] = lnk[u] = ++time;                7e41
        s.push(u);                               80f6
        for (int v : adj[u]){                    18f6
            if (!pre[v]){                        173e
                dfs(v);                          5f3c
                lnk[u] = min(lnk[u], lnk[v]);    002c
            } else if (!scc[v]){                 6068
                lnk[u] = min(lnk[u], pre[v]);    d5df
```

```
            }
        }
        if (lnk[u] == pre[u]){
            sccn++;
            int x;
            do {
                x = s.top(); s.pop();
                scc[x] = sccn;
            } while (x != u);
        }
    }

    void find_scc(){
        time = sccn = 0;
        memset(scc, 0, sizeof scc);
        memset(pre, 0, sizeof pre);
        Rep (i, V){
            if (!pre[i]) dfs(i);
        }
    }

    vector<int> adjc[MAXV];
    void contract(){
        Rep (i, V)
            rep (j, adj[i].size()){
                if (scc[i] != scc[adj[i][j]])
                    adjc[scc[i]].push_back(scc[adj[i][j]]);
            }
    }
};
```

5.2 Vertex biconnected component

```
const int MAXN = 100005;
struct graph {
    int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
    vector<int> adj[MAXN], bcc[MAXN];
    set<pair<int, int>> bcce[MAXN];

    stack<pair<int, int>> s;

    void add_edge(int u, int v) {
```

```
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int dfs(int u, int fa) {
        int lowu = pre[u] = ++dfs_clock;
        int child = 0;
        for (int v : adj[u]) {
            if (!pre[v]) {
                s.push({u, v});
                child++;
                int lowv = dfs(v, u);
                lowu = min(lowu, lowv);
                if (lowv >= pre[u]) {
                    iscut[u] = 1;
                    bcc[bcc_cnt].clear();
                    bcce[bcc_cnt].clear();
                    while (1) {
                        int xu, xv;
                        tie(xu, xv) = s.top(); s.pop();
                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
                        if (bccno[xu] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xu);
                            bccno[xu] = bcc_cnt;
                        }
                        if (bccno[xv] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xv);
                            bccno[xv] = bcc_cnt;
                        }
                        if (xu == u && xv == v) break;
                    }
                    bcc_cnt++;
                }
            } else if (pre[v] < pre[u] && v != fa) {
                s.push({u, v});
                lowu = min(lowu, pre[v]);
            }
        }
        if (fa < 0 && child == 1) iscut[u] = 0;
        return lowu;
    }

    void find_bcc(int n) {
        memset(pre, 0, sizeof pre);
```

```
e2d2         memset(iscut, 0, sizeof iscut);
40d3         memset(bccno, -1, sizeof bccno);
fae2         dfs_clock = bcc_cnt = 0;
5c63         rep (i, n) if (!pre[i]) dfs(i, -1);
95cf     }
329b };
```

5.3 Cut vertices

If the graph is unconnected, the algorithm should be run on each component.
Usage:

`tarjan(u, fa)`	Run Tarjan's algorithm on tree rooted at `fa`. Please call with identical `u` and `fa`.

```
9f60 const int MAXN = 200005;
0b32 vector<int> adj[MAXN];
18e4 int dfn[MAXN], low[MAXN], idx;
d39d bool cut[MAXN];
427e
bfab void add_edge(int u, int v) {
c71a     adj[u].push_back(v);
a717     adj[v].push_back(u);
95cf }
427e
50aa void tarjan(int u, int fa) {
9891     dfn[u] = low[u] = ++idx;
ec14     int child = 0;
18f6     for (int v : adj[u]) {
3c64         if (!dfn[v]) {
9636             tarjan(v, fa); low[u] = min(low[u], low[v]);
f368             if (low[v] >= dfn[u] && u != fa) cut[u] = true;
7923             child += u == fa;
95cf         }
769a         low[u] = min(low[u], dfn[v]);
95cf     }
7927     if (u == fa && child > 1) cut[u] = true;
95cf }
```

5.4 Minimum spanning arborescence (Chu-Liu)

All vertices are 1-based.
Usage:

`getans(n, root, edges)`	Compute the total weight of MSA rooted at `root`.

Time Complexity: $O(|V||E|)$

```
struct edge {                                                           bcf8
    int u, v;                                                           54f1
    LL w;                                                               309c
};                                                                      329b
                                                                        427e
const int MAXN = 10005;                                                 f5a4
LL in[MAXN];                                                            7124
int pre[MAXN], vis[MAXN], id[MAXN];                                     1c1d
                                                                        427e
LL getans(int n, int rt, vector<edge>& edges) {                         5a43
    LL ans = 0;                                                         f7ff
    int cnt = 0;                                                        8abb
    while (1) {                                                         a147
        Rep (i, n) in[i] = LLONG_MAX, id[i] = vis[i] = 0;               641a
        for (auto e : edges) {                                          0705
            if (e.u != e.v and e.w < in[e.v]) {                         073a
                pre[e.v] = e.u;                                         c1df
                in[e.v] = e.w;                                          5fbc
            }                                                           95cf
        }                                                               95cf
        in[rt] = 0;                                                     3fdb
        Rep (i, n) {                                                    34d7
            if (in[i] == LLONG_MAX) return -1;                          3c97
            ans += in[i];                                               cf57
            int u;                                                      a763
            for (u = i; u != rt && vis[u] != i && !id[u]; u = pre[u])   4b0e
                vis[u] = i;                                             88a2
            if (u != rt && !id[u]) {                                    4b22
                id[u] = ++cnt;                                          b66e
                for (int v = pre[u]; v != u; v = pre[v])                0443
                    id[v] = cnt;                                        5c22
            }                                                           95cf
        }                                                               95cf
        if (!cnt) return ans;                                           91e9
        Rep (i, n) if (!id[i]) id[i] = ++cnt;                           5e22
        for (auto& e : edges) {                                         7400
            LL laz = in[e.v];                                           7750
            e.u = id[e.u];                                              97ae
            e.v = id[e.v];                                              fae6
            if (e.u != e.v) e.w -= laz;                                 bdd2
```

```
95cf         }
6cc4         n = cnt; rt = id[rt]; cnt = 0;
95cf     }
95cf }
```

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

```
5ece const int MAXN = 300005;
2fef typedef pair<LL, int> pii;
1495 struct MDST {
01b2     priority_queue<pii, vector<pii>, greater<pii>> heap[MAXN];
321d     LL shift[MAXN];
fc06     int fa[MAXN], vis[MAXN];
427e 
38dd     int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
427e 
29b0     void unite(int x, int y) {
0c14         x = find(x); y = find(y); fa[y] = x; if (x == y) return;
6fa0         if (heap[x].size() < heap[y].size()) {
9c26             swap(heap[x], heap[y]);
2ffc             swap(shift[x], shift[y]);
95cf         }
9959         while (heap[y].size()) {
175b             auto p = heap[y].top(); heap[y].pop();
c353             heap[x].emplace(p.first + shift[y] - shift[x], p.second);
95cf         }
95cf     }
427e 
0bbd     void add_edge(int u, int v, LL w) { heap[v].emplace(w, u); }
427e 
a526     LL run(int n, int rt) {
f7ff         LL ans = 0;
81f2         iota(fa, fa + n + 1, 0);
19b3         Rep (i, n) if (find(i) != find(rt)) {
a7b1             int u = find(i);
                 stack<int, vector<int>> s;                                      010e
                 while (find(u) != find(rt)) {                                   eff5
                     if (vis[u]) while (s.top() != u) {                          0dda
                         vis[s.top()] = 0; unite(u, s.top()); s.pop();           c593
                     } else { vis[u] = 1; s.push(u); }                           83c4
                     while (heap[u].size()) {                                    c76e
                         ans += heap[u].top().first + shift[u];                  2077
                         shift[u] = -heap[u].top().first;                        4905
                         if (find(heap[u].top().second) != u) break;             da47
                         heap[u].pop();                                          9fbb
                     }                                                           95cf
                     if (heap[u].empty()) return LLONG_MIN;                      6961
                     u = find(heap[u].top().second);                             87e6
                 }                                                               95cf
                 while (s.size()) { vis[s.top()] = 0; unite(rt, s.top()); s.pop(); }  2d46
             }                                                                   95cf
             return ans;                                                         4206
         }                                                                       95cf
     };                                                                          329b
```

5.6 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, $O(\sqrt{V}E)$.

```
struct edge{                                          bcf8
    int from, to;                                     60e2
    LL cap, flow;                                     5e6d
};                                                    329b
                                                      427e
const int MAXN = 1005;                                e2cd
struct Dinic {                                        9062
    int n, m, s, t;                                   4dbf
    vector<edge> edges;                               9f0c
    vector<int> G[MAXN];                              b891
    bool vis[MAXN];                                   bbb6
    int d[MAXN];                                      b40a
    int cur[MAXN];                                    ddec
                                                      427e
    void add_edge(int from, int to, LL cap) {         5973
```

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

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

5.7 Maximum cardinality bipartite matching (Hungarian)

```
#include <bits/stdc++.h>                                               302f
using namespace std;                                                   421c
                                                                       427e
#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) (x).begin(), (x).end()                                8843
typedef long long LL;                                                  5cad
                                                                       427e
struct Hungarian{                                                      84ee
    int nx, ny;                                                        fbf6
    vector<int> mx, my;                                                9ec6
    vector<vector<int> > e;                                            9d4c
    vector<bool> mark;                                                 edec
                                                                       427e
    void init(int nx, int ny){                                         8324
        this->nx = nx;                                                 c1d1
        this->ny = ny;                                                 f9c1
        mx.resize(nx); my.resize(ny);                                  ac92
        e.clear(); e.resize(nx);                                       3f11
        mark.resize(nx);                                               1023
```

```cpp
    }

    inline void add(int a, int b){
        e[a].push_back(b);
    }

    bool augment(int i){
        if (!mark[i]) {
            mark[i] = true;
            for (int j : e[i]){
                if (my[j] == -1 || augment(my[j])){
                    mx[i] = j; my[j] = i;
                    return true;
                }
            }
        }
        return false;
    }

    int match(){
        int ret = 0;
        fill(range(mx), -1);
        fill(range(my), -1);
        rep (i, nx){
            fill(range(mark), false);
            if (augment(i)) ret++;
        }
        return ret;
    }
};
```

5.8 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.

```cpp
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);
```

```
95cf            }
95cf        }
95cf    }
427e
a0ce    int solve() {
34d7        Rep (i, n) {
a073            if (mate[i]) continue;
1fc0            Rep (j, n) label[j] = -1;
7676            label[i] = 0; q = queue<int>(); q.push(i);
1c7d            while (q.size()) {
66ba                int x = q.front(); q.pop();
b98c                for (int y : adj[x]) {
c07f                    if (mate[y] == 0 and i != y) {
7f36                        mate[y] = x; rematch(x, y); q = queue<int>(); break;
95cf                    }
d315                    if (label[y] >= 0) { relabel(x, y); continue; }
58ec                    if (label[mate[y]] < 0) {
c9c4                        label[mate[y]] = x; q.push(mate[y]);
95cf                    }
95cf                }
95cf            }
95cf        }
8abb        int cnt = 0;
b52f        Rep (i, n) cnt += (mate[i] > i);
6808        return cnt;
95cf    }
329b };
```

5.9 Minimum cost maximum flow

```
bcf8 struct edge{
60e2     int from, to;
d698     int cap, flow;
32cc     LL cost;
329b };
427e
cc3e const LL INF = LLONG_MAX / 2;
2aa8 const int MAXN = 5005;
c6cb struct MCMF {
9ceb     int s, t, n, m;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
    bool inq[MAXN]; // queue                                              f74f
    LL d[MAXN];     // distance                                           8f67
    int p[MAXN];    // previous                                           9524
    int a[MAXN];    // improvement                                        b330
                                                                          427e
    void add_edge(int from, int to, int cap, LL cost) {                   f7f2
        edges.push_back(edge{from, to, cap, 0, cost});                    24f0
        edges.push_back(edge{to, from, 0, 0, -cost});                     95f0
        m = edges.size();                                                 fe77
        G[from].push_back(m-2);                                           dff5
        G[to].push_back(m-1);                                             8f2d
    }                                                                     95cf
                                                                          427e
    bool spfa(){                                                          3c52
        queue<int> q;                                                     93d2
        fill(d, d + MAXN, INF); d[s] = 0;                                 8494
        memset(inq, 0, sizeof(inq));                                      fd48
        q.push(s); inq[s] = true;                                         5e7c
        p[s] = 0; a[s] = INT_MAX;                                         2dae
        while (!q.empty()){                                               cc78
            int u = q.front(); q.pop(); inq[u] = false;                   b0aa
            for (int i : G[u]) {                                          3bba
                edge& e = edges[i];                                       56d8
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost){           3601
                    d[e.to] = d[u] + e.cost;                              55bc
                    p[e.to] = G[u][i];                                    0bea
                    a[e.to] = min(a[u], e.cap - e.flow);                  8249
                    if (!inq[e.to]) q.push(e.to), inq[e.to] = true;       e5d3
                }                                                         95cf
            }                                                             95cf
        }                                                                 95cf
        return d[t] != INF;                                               6d7c
    }                                                                     95cf
                                                                          427e
    void augment(){                                                       71a4
        int u = t;                                                        06f1
        while (u != s){                                                   b19d
            edges[p[u]].flow += a[t];                                     db09
            edges[p[u]^1].flow -= a[t];                                   25a9
            u = edges[p[u]].from;                                         e6c9
        }                                                                 95cf
    }                                                                     95cf
                                                                          427e
#ifdef GIVEN_FLOW                                                         6e20
```

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

5.10 Global minimum cut (Stoer-Wagner)

Usage:

`stoer(w)`	Compute the global minimum cut of the graph specified by the **symmetric** adjacenct matrix `w` (0-based). Return the capaity of the cut and the indices of one part of the cut.

Time Complexity: $O(|V|^3)$

```
f9d7  typedef vector<LL> VI;
045e  typedef vector<VI> VVI;
427e
f012  pair<LL, VI> stoer(VVI &w) {
66f7      int n = w.size();
```

```
    VI used(n), c, bestc;                                                  4d98
    LL bestw = -1;                                                         329d
                                                                           427e
    for (int ph = n - 1; ph >= 0; ph--) {                                  cd21
        VI wt = w[0], added = used;                                        ec6e
        int prev, last = 0;                                                f20e
        rep (i, ph) {                                                      4b32
            prev = last;                                                   8bfc
            last = -1;                                                     0706
            for (int j = 1; j < n; j++)                                    4942
                if (!added[j] && (last == -1 || wt[j] > wt[last]))         c4b9
                    last = j;                                              887d
            if (i == ph - 1) {                                             71bc
                rep (j, n) w[prev][j] += w[last][j];                       9cfa
                rep (j, n) w[j][prev] = w[prev][j];                        1f25
                used[last] = true;                                         5613
                c.push_back(last);                                         8e11
                if (bestw == -1 || wt[last] < bestw) {                     bb8e
                    bestc = c;                                             bab6
                    bestw = wt[last];                                      372e
                }                                                          95cf
            } else {                                                       8e2e
                rep (j, n) wt[j] += w[last][j];                            caeb
                added[last] = true;                                        8b92
            }                                                              95cf
        }                                                                  95cf
    }                                                                      95cf
    return {bestw, bestc};                                                 038c
}                                                                          95cf
```

5.11 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
```

```
0df2        st[id[u] = nid++][0] = {d, u};
18f6        for (int v : adj[u]) {
bd87            if (v == p) continue;
f58c            dfs(v, u, d + 1);
08ad            st[nid++][0] = {d, u};
95cf        }
95cf    }
427e
3d1b    void preprocess(int root) {
3269        nid = 0;
91e1        dfs(root, 0, 1);
5e98        int l = 31 - __builtin_clz(nid);
213b        rep (j, l) rep (i, 1+nid-(1<<j))
1131            st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
95cf    }
427e
0f0b    int lca(int u, int v) {
cfc4        tie(u, v) = minmax(id[u], id[v]);
be9b        int k = 31 - __builtin_clz(v-u+1);
8ebc        return min(st[u][k], st[v-(1<<k)+1][k]).second;
95cf    }
```

5.12 Heavy-light decomposition

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

```
0f42    const int MAXN = 100005;
0b32    vector<int> adj[MAXN];
42f2    int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];
427e
be5c    void dfs1(int x, int dep, int par){
7489        depth[x] = dep;
2ee7        sz[x] = 1;
adb4        fa[x] = par;
b79d        int maxn = 0, s = 0;
c861        for (int c: adj[x]){
fe45            if (c == par) continue;
fd2f            dfs1(c, dep + 1, x);
b790            sz[x] += sz[c];
f0f1            if (sz[c] > maxn){
c749                maxn = sz[c];
fe19                s = c;
            }                                                       95cf
        }                                                           95cf
        son[x] = s;                                                 0e08
    }                                                               95cf
                                                                    427e
    int cid = 0;                                                    ba54
    void dfs2(int x, int t){                                        3644
        top[x] = t;                                                 8d96
        id[x] = ++cid;                                              d314
        if (son[x]) dfs2(son[x], t);                                c4a1
        for (int c: adj[x]){                                        c861
            if (c == fa[x]) continue;                               9881
            if (c == son[x]) continue;                              5518
            else dfs2(c, c);                                        13f9
        }                                                           95cf
    }                                                               95cf
                                                                    427e
    void decomp(int root){                                          0f04
        dfs1(root, 1, 0);                                           9fa4
        dfs2(root, root);                                           1c88
    }                                                               95cf
                                                                    427e
    void query(int u, int v){                                       2c98
        while (top[u] != top[v]){                                   03a1
            if (depth[top[u]] < depth[top[v]]) swap(u, v);          45ec
            // id[top[u]] to id[u]                                  427e
            u = fa[top[u]];                                         005b
        }                                                           95cf
        if (depth[u] > depth[v]) swap(u, v);                        6083
        // id[u] to id[v]                                           427e
    }                                                               95cf
```

5.13 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.

```cpp
vector<int> adj[100005];
int sz[100005], sum;

void getsz(int u, int p) {
  sz[u] = 1; sum++;
  for (int v : adj[u]) {
    if (v == p) continue;
    getsz(v, u);
    sz[u] += sz[v];
  }
}

int getcent(int u, int p) {
  for (int v : adj[u])
    if (v != p and sz[v] > sum / 2)
      return getcent(v, u);
  return u;
}

void decompose(int u) {
  sum = 0; getsz(u, 0);
  u = getcent(u, 0); // update u to the centroid

  for (int v : adj[u]) {
    // get answer for subtree v
  }
  // get answer for the whole tree
  // don't forget to count the centroid itself

  for (int v : adj[u]) { // divide and conquer
    adj[v].erase(find(range(adj[v]), u));
    decompose(v);
    adj[v].push_back(u); // restore deleted edge
  }
}
```

5.14 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`.

```cpp
vector<int> adj[100005];
int sz[100005], son[100005];

void decomp(int u, int p) {
    sz[u] = 1;
    for (int v : adj[u]) {
        if (v == p) continue;
        decomp(v, u);
        sz[u] += sz[v];
        if (sz[v] > sz[son[u]]) son[u] = v;
    }
}

template <typename T>
void trav(T fn, int u, int p) {
    fn(u);
    for (int v : adj[u]) if (v != p) trav(fn, v, u);
}

#define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
void work(int u, int p, bool keep) {
    for_light(v) work(v, u, 0); // process light children

    // process heavy child
    // current data structure contains info of heavy child
    if (son[u]) work(son[u], u, 1);

    auto merge = [u] (int c) { /* count contribution of c */ };
    auto enter = [] (int c) { /* add vertex c */ };
    auto leave = [] (int c) { /* remove vertex c*/ };

    for_light(v) {
        trav(merge, v, u);
        trav(enter, v, u);
    }

    // count answer for root and add it
    // Warning: special check may apply to root!
```

```
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
d34f     void init(int n) { // fill the array with 0
1010         tr.resize(N = n + 5);
95cf     }
427e
63d0     LL sum(int n) {
f7ff         LL ans = 0;
e290         while (n) {
0715             ans += tr[n];
c0d4             n &= n - 1;
95cf         }
4206         return ans;
95cf     }
427e
f4bd     void add(int n, LL x){
ad20         while (n < N) {
6c81             tr[n] += x;
0af5             n += n & -n;
95cf         }
95cf     }
329b };
```

6.2 Fenwick tree (range update point query)

```
3d03 struct bit_rupq{ // range update, point query
d7af     int N;
    vector<LL> tr;                                  99ff
                                                    427e
    void init(int n) { // fill the array with 0     d34f
        tr.resize(N = n + 5);                       1010
    }                                               95cf
                                                    427e
    LL query(int n) {                               38d4
        LL ans = 0;                                 f7ff
        while (n < N) {                             ad20
            ans += tr[n];                           0715
            n += n & -n;                            0af5
        }                                           95cf
        return ans;                                 4206
    }                                               95cf
                                                    427e
    void add(int n, LL x) {                         f4bd
        while (n){                                  e290
            tr[n] += x;                             6c81
            n &= n - 1;                             c0d4
        }                                           95cf
    }                                               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
```

```

  void push_mul(int o, LL x) {
    val[o] = val[o] * x % p;
    tadd[o] = tadd[o] * x % p;
    tmul[o] = tmul[o] * x % p;
  }

  void push(int o) {
    if (l[o] == m[o]) return;
    if (tmul[o] != 1) {
      push_mul(lson, tmul[o]);
      push_mul(rson, tmul[o]);
      tmul[o] = 1;
    }
    if (tadd[o]) {
      push_add(lson, tadd[o]);
      push_add(rson, tadd[o]);
      tadd[o] = 0;
    }
  }

  void build(int o, int ll, int rr) {
    int mm = (ll + rr) / 2;
    l[o] = ll; r[o] = rr; m[o] = mm;
    tmul[o] = 1;
    if (ll == mm) {
      scanf("%lld", val + o);
      val[o] %= p;
    } else {
      build(lson, ll, mm);
      build(rson, mm, rr);
      pull(o);
    }
  }

  void add(int o, int ll, int rr, LL x) {
    if (ll <= l[o] && r[o] <= rr) {
      push_add(o, x);
    } else {
      push(o);
      if (m[o] > ll) add(lson, ll, rr, x);
      if (m[o] < rr) add(rson, ll, rr, x);
      pull(o);
    }
  }

  void mul(int o, int ll, int rr, LL x) {
    if (ll <= l[o] && r[o] <= rr) {
      push_mul(o, x);
    } else {
      push(o);
      if (ll < m[o]) mul(lson, ll, rr, x);
      if (m[o] < rr) mul(rson, ll, rr, x);
      pull(o);
    }
  }

  LL query(int o, int ll, int rr) {
    if (ll <= l[o] && r[o] <= rr) {
      return val[o];
    } else {
      push(o);
      if (rr <= m[o]) return query(lson, ll, rr);
      if (ll >= m[o]) return query(rson, ll, rr);
      return query(lson, ll, rr) + query(rson, ll, rr);
    }
  }
} seg;
```

6.4 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;
```

```
a7c5 mt19937 gen(time(NULL));
9542 struct Treap {
6d61     int ch[MAXN][2];
3948     int sz[MAXN], key[MAXN], val[MAXN];
5d9a     int add[MAXN], rev[MAXN];
2b1b     LL sum[MAXN] = {0};
a773     int maxv[MAXN] = {INT_MIN}, minv[MAXN] = {INT_MAX};
427e
a629     void Init(int x, int v) {
5a00         ch[x][0] = ch[x][1] = 0;
d8cd         key[x] = gen(); val[x] = v; pull(x);
95cf     }
427e
3bf9     void pull(int x) {
e1c3         sz[x] = 1 + sz[ch[x][0]] + sz[ch[x][1]];
99f8         sum[x] = val[x] + sum[ch[x][0]] + sum[ch[x][1]];
94e9         maxv[x] = max({val[x], maxv[ch[x][0]], maxv[ch[x][1]]});
6bb9         minv[x] = min({val[x], minv[ch[x][0]], minv[ch[x][1]]});
95cf     }
427e
8c8e     void Add(int x, int a) {
a7b1         val[x] += a; add[x] += a;
832a         sum[x] += LL(sz[x]) * a; maxv[x] += a; minv[x] += a;
95cf     }
427e
aaf6     void Reverse(int x) {
52c6         rev[x] ^= 1;
7850         swap(ch[x][0], ch[x][1]);
95cf     }
427e
1a53     void push(int x) {
5fe5         for (int c : ch[x]) if (c) {
fd76             Add(c, add[x]);
7a53             if (rev[x]) Reverse(c);
95cf         }
49ee         add[x] = 0; rev[x] = 0;
95cf     }
427e
9d2c     int Merge(int x, int y) {
1b09         if (!x || !y) return x | y;
cd7e         push(x); push(y);
bffa         if (key[x] > key[y]) {
a3df             ch[x][1] = Merge(ch[x][1], y); pull(x); return x;
8e2e         } else {
            ch[y][0] = Merge(x, ch[y][0]); pull(y); return y; bf9e
        } 95cf
    } 95cf
 427e
    void Split(int t, int k, int &x, int &y) { dc7e
        if (t == 0) { x = y = 0; return; } 6303
        push(t); f26b
        if (sz[ch[t][0]] < k) { 3465
            x = t; Split(ch[t][1], k - sz[ch[t][0]] - 1, ch[t][1], y); ffd8
        } else { 8e2e
            y = t; Split(ch[t][0], k, x, ch[t][0]); 8a23
        } 95cf
        if (x) pull(x); if (y) pull(y); 89e3
    } 95cf
} treap; b1f4
 427e
int root; 24b6
 427e
void init(int n) { d34f
    Rep (i, n) { 34d7
        int x; scanf("%d", &x); 7681
        treap.Init(i, x); 0ed8
        root = (i == 1) ? 1 : treap.Merge(root, i); bcc8
    } 95cf
} 95cf
 427e
void work(int op, int l, int r) { d030
    int tl, tm, tr; 6639
    treap.Split(root, l, tl, tm); b6c4
    treap.Split(tm, r - l, tm, tr); 8de3
    if (op == 1) { 3658
        int x; scanf("%d", &x); treap.Add(tm, x); c039
    } else if (op == 2) { 1dcb
        treap.Reverse(tm); ae78
    } else if (op == 3) { 581d
        printf("%lld␣%d␣%d\n", e092
                treap.sum[tm], treap.minv[tm], treap.maxv[tm]); 867f
    } 95cf
    root = treap.Merge(treap.Merge(tl, tm), tr); 6188
} 95cf
```

6.5 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;
6d09         fa[y] = x; fa[x] = z; pull(y);
95cf     }
52c6     void pushall(int x) { if (isroot(x)) pushall(fa[x]); push(x); }
f69c     void splay(int x) {
d095         int y = x, z = 0;
c494         for (pushall(y); isroot(x); rotate(x)) {
ceef             y = fa[x]; z = fa[y];
4449             if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
95cf         }
78a0         pull(x);
95cf     }
6229     void access(int x) {
```

```
        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
```

6.6 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
// SAMPLE USAGE                                                                 427e
rkt.insert(x);          // insert element                                       190e
rkt.erase(x);           // erase element                                        05d4
rkt.order_of_key(x);    // obtain the number of elements less than x            add5
rkt.find_by_order(i);   // iterator to i-th (numbered from 0) smallest element  b064
rkt.lower_bound(x);                                                             c103
rkt.upper_bound(x);                                                             4ff4
rkt.join(rkt2);         // merge tree (only if their ranges do not intersect)   b19b
rkt.split(x, rkt2);     // split all elements greater than x to rkt2            cb47
```

6.7 Persistent segment tree, range k-th query

```
struct node {
  static int n, pos;

  int value;
  node *left, *right;

  void* operator new(size_t size);

  static node* Build(int l, int r) {
    node* a = new node;
    if (r > l + 1) {
      int mid = (l + r) / 2;
      a->left = Build(l, mid);
      a->right = Build(mid, r);
    } else {
      a->value = 0;
    }
    return a;
  }

  static node* init(int size) {
    n = size;
    pos = 0;
    return Build(0, n);
  }

  static int Query(node* lt, node *rt, int l, int r, int k) {
    if (r == l + 1) return l;
    int mid = (l + r) / 2;
    if (rt->left->value - lt->left->value < k) {
      k -= rt->left->value - lt->left->value;
      return Query(lt->right, rt->right, mid, r, k);
    } else {
      return Query(lt->left, rt->left, l, mid, k);
    }
  }

  static int query(node* lt, node *rt, int k) {
    return Query(lt, rt, 0, n, k);
  }

  node *Inc(int l, int r, int pos) const {
    node* a = new node(*this);
    if (r > l + 1) {
      int mid = (l + r) / 2;
      if (pos < mid)
        a->left = left->Inc(l, mid, pos);
      else
        a->right = right->Inc(mid, r, pos);
    }
    a->value++;
    return a;
  }

  node *inc(int index) {
    return Inc(0, n, index);
  }
} nodes[8000000];

int node::n, node::pos;
inline void* node::operator new(size_t size) {
  return nodes + (pos++);
}
```

6.8 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.

```
const int BLOCK = 800;
typedef vector<int> vi;

struct block {
    vi data;
    LL sum; int minv, maxv;
    int add; bool rev;

    block(vi&& vec) : data(move(vec)),
        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
7b8e     void maintain() {
3131         lit it = blk.begin();
4628         while (it != blk.end() && next(it) != blk.end()) {
852d             lit it2 = it;
188c             while (next(it2) != blk.end() &&
3600                     it2->data.size() + next(it2)->data.size() <= BLOCK) {
93e1                 it2->merge(*next(it2));
e1fa                 blk.erase(next(it2));
95cf             }
5771             ++it;
95cf         }
95cf     }
427e
b7b3     lit split(int pos) {
2273         for (lit it = blk.begin(); ; it++) {
```

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

```
95cf     }
958e } lst;
```

6.9 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 {
2989     pcvi data;
8fd0     LL sum;
427e
427e     // add information to maintain
a613     block(pcvi ptr) :
24b5         data(ptr),
0cf0         sum(accumulate(ptr->begin(), ptr->end(), 0ll))
e93b     { }
427e
5c0f     void merge(const block& another) {
0b18         pvi temp = make_shared<vi>(data->begin(), data->end());
ac21         temp->insert(temp->end(), another.data->begin(), another.data->end());
6467         *this = block(temp);
95cf     }
427e
42e8     block split(int pos) {
dac1         block result(make_shared<vi>(data->begin() + pos, data->end()));
01db         *this = block(make_shared<vi>(data->begin(), data->begin() + pos));
56b0         return result;
95cf     }
329b };
427e
2a18 typedef list<block>::iterator lit;
427e
ce14 struct blocklist {
5540     list<block> blk;
427e
7b8e     void maintain() {
3131         lit it = blk.begin();
5e44         while (it != blk.end() and next(it) != blk.end()) {
852d             lit it2 = it;
0b03             while (next(it2) != blk.end() and
029f                     it2->data->size() + next(it2)->data->size() <= BLOCK) {
93e1                 it2->merge(*next(it2));
e1fa                 blk.erase(next(it2));
95cf             }
5771             ++it;
95cf         }
95cf     }
427e
b7b3     lit split(int pos) {
2273         for (lit it = blk.begin(); ; it++) {
5502             if (pos == 0) return it;
d480             while (it->data->size() > pos) {
2099                 blk.insert(next(it), it->split(pos));
95cf             }
a1c8             pos -= it->data->size();
95cf         }
95cf     }
427e
fd38     LL sum(int l, int r) { // traverse
48b4         lit it1 = split(l), it2 = split(r);
ac09         LL res = 0;
9f1d         while (it1 != it2) {
8284             res += it1->sum;
61fd             it1++;
95cf         }
b204         maintain();
244d         return res;
95cf     }
```

```
};
```

6.10 Sparse table, range extremum query

The array is 0-based and the range is closed.

```
const int MAXN = 100007;
int a[MAXN];
int st[MAXN][32 - __builtin_clz(MAXN)];

inline int ext(int x, int y){return x>y?x:y;} // ! max

void init(int n){
    int l = 31 - __builtin_clz(n);
    rep (i, n) st[i][0] = a[i];
    rep (j, l)
        rep (i, 1+n-(1<<j))
            st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
}

int rmq(int l, int r){
    int k = 31 - __builtin_clz(r-l+1);
    return ext(st[l][k], st[r-(1<<k)+1][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
inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}

inline bool intersect(seg a, seg b){
    // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
      and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
           rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
}

// 0 not insersect
// 1 standard intersection
// 2 vertex-line intersection
// 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     }
0a5f     return wn ? 1 : -1;
95cf }
427e
d4a3 istream& operator >> (istream& lhs, pt& rhs){
fa86     lhs >> rhs.x >> rhs.y;
331a     return lhs;
95cf }
427e
07ae istream& operator >> (istream& lhs, seg& rhs){
5cab     lhs >> rhs.first >> rhs.second;
331a     return lhs;
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 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 polyadd(a, b) : return map(lambda x, y : (x or 0) + (y or 0), a, b)    bbbe
                                                                            427e
def polymul(a, b) :                                                         796b
    p = [0] * (len(a)+len(b)-1)                                             83e4
    for e1, c1 in enumerate(a) :                                            f697
```

```
156c          for e2, c2 in enumerate(b) :
dfce              p[e1 + e2] += c1 * c2
5849      return p
427e
f06d  x, y = [map(Fraction, raw_input().split()) for _ in 0,0]
e80a  n = len(x)
a649  lj = [reduce(polymul, [[-x[m]/(x[j]-x[m]), 1/(x[j]-x[m])]
9dfa      for m in range(n) if m != j]) for j in range(n)]
46f9  print '␣'.join(map(str, reduce(polyadd,
d754      map(lambda a, b : [x * a for x in b], y, lj))))
```