南京大学 ACM-ICPC 集训队代码模版库

Linux-4.15.0-70-generic-x86_64-with-Ubuntu-18.04-bionic
XeTeX 3.14159265-2.6-0.99998 (TeX Live 2017/Debian)
CPython 2.7.15+
2019-12-06 15:09:55.231114, build 0072

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

Usage:

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

```
0f42 const int MAXN = 100005;
03a9 struct twoSAT {
5c83     int n;
8f72     vector<int> G[MAXN*2];
d060     bool mark[MAXN*2];
b42d     int S[MAXN*2], c;
427e
d34f     void init(int n) {
b985         this->n = n;
f9ec         for (int i=0; i < n*2; i++) G[i].clear();
0609         memset(mark, 0, sizeof(mark));
95cf     }
427e
3bd5     bool dfs(int x) {
bd70         if (mark[x^1]) return false;
c96a         if (mark[x]) return true;
fd23         mark[x] = true;
4bea         S[c++] = x;
bd55         for (int u : G[x]) if (!dfs(u)) return false;
3361         return true;
95cf     }
427e
5894     void add_clause(int x, bool xval, int y, bool yval) {
6afe         x = x * 2 + xval;
e680         y = y * 2 + yval;
2be7         G[x].push_back(y);
95cf     }
427e
d0cb     bool solve() {
7c39         for (int i=0; i<n*2; i+=2) {
e63f             if (!mark[i] && !mark[i+1]) {
88fb                 c = 0;
```

```
                if (!dfs(i)) {                                  f4b9
                    while (c > 0) mark[S[--c]] = false;         3f03
                    if (!dfs(i+1)) return false;                86c5
                }                                               95cf
            }                                                   95cf
        }                                                       95cf
        return true;                                            3361
    }                                                           95cf
                                                                427e
    bool operator[] (int x) { return mark[2*x+1]; }             fb3b
};                                                              329b
```

2.2 Matroid Intersection

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

Usage:

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

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

```cpp

    vector<int> getcur() {
        vector<int> ret;
        rep (i, n) if (in[i]) ret.push_back(i);
        return ret;
    }

    void enqueue(int x, int p) {
        if (vis[x]) return;
        vis[x] = true; pre[x] = p; q.push(x);
        if (t[x]) throw x;
    };

    vector<int> run() {
        while (true) {
            vector<int> cur = getcur();
            fill(vis, vis + n, 0);
            rep (i, n) adj[i].clear();
            MT2 mt2;
            for (int i : cur) mt2.insert(i);
            rep (i, n) t[i] = mt2.test(i);
            vector<MT1> mt1s(cur.size());
            vector<MT2> mt2s(cur.size());
            rep (i, cur.size()) rep (j, cur.size()) if (i != j) {
                mt1s[i].insert(cur[j]);
                mt2s[i].insert(cur[j]);
            }
            rep (i, n) if (!in[i]) rep (j, cur.size()) {
                if (mt1s[j].test(i)) adj[cur[j]].push_back(i);
                if (mt2s[j].test(i)) adj[i].push_back(cur[j]);
            }
            q = {};
            try {
                MT1 mt1;
                for (int i : cur) mt1.insert(i);
                rep (i, n) if (mt1.test(i)) enqueue(i, -1);
                while (q.size()) {
                    int u = q.front(); q.pop();
                    for (int v : adj[u]) enqueue(v, u);
                }
            } catch (int v) {
                while (v >= 0) { in[v] ^= 1; v = pre[v]; }
                continue;
            }
                break;
        };
        return getcur();
    }
};
```

2.3 Connectivity Dynamic Programming

```cpp
const ULL WIDTH = 3, MASK = (1 << WIDTH) - 1, CONN = 1;
int n, m;

ULL Get(ULL mask, int digit) {
    return (mask >> (digit * WIDTH)) & MASK;
}

[[gnu::warn_unused_result]]
ULL Set(ULL mask, int digit, ULL val) {
    digit *= WIDTH;
    return (mask & ~(MASK << digit)) | val << digit;
}

[[gnu::warn_unused_result]]
ULL Set(ULL mask, int digit, ULL val1, ULL val2) {
    return mask = Set(mask, digit, val1), Set(mask, digit+1, val2);
}

ULL Canon(ULL mask) {
    ULL repr[1 << WIDTH] = {}, top = CONN;
    rep (i, m + 1) {
        ULL val = Get(mask, i);
        if (val < CONN) continue;
        if (repr[val] == 0) repr[val] = top++;
        mask = Set(mask, i, repr[val]);
    }
    return mask;
}

ULL Unite(LL mask, ULL val1, ULL val2) {
    rep (i, m + 1) if (Get(mask, i) == val1) mask = Set(mask, i, val2);
    return Canon(mask);
}

```

```
1853 char g[16][16];
1203 unordered_map<ULL, ULL> dp[16][16];
427e
3117 int main() {
d6ef     fgets(g[0], sizeof(g[0]), stdin);
4ae7     sscanf(g[0], "%d %d", &n, &m);
454b     rep (i, n) fgets(g[i], sizeof(g[i]), stdin);
4873     int lasti = n, lastj;
8a11     while (lasti--) for (lastj = m; lastj; lastj--)
00ff         if (g[lasti][lastj-1] == '.') goto cont;
9c8f     cont:;
be8e     rep (i, n) {
d8e8         if (i) for (auto pr : dp[i-1][m]) {
a8a5             ULL mask, val; tie(mask, val) = pr;
7d60             if (Get(mask, m) == 0) dp[i][0][mask << WIDTH] += val;
8e2e         } else {
664e             dp[0][0][0] = 1;
95cf         }
1fc5         rep (j, m) for (auto pr : dp[i][j]) {
a8a5             ULL mask, val; tie(mask, val) = pr;
289a             ULL d1 = Get(mask, j), d2 = Get(mask, j + 1);
ab58             if (g[i][j] == '.') {
9625                 if (d1 == 0 and d2 == 0) {
cac2                     dp[i][j+1][Canon(Set(mask, j, MASK, MASK))] += val;
c909                 } else if (d1 == 0 or d2 == 0) {
a611                     dp[i][j+1][mask] += val;
4349                     mask = Set(mask, j, d2); mask = Set(mask, j + 1, d1);
a611                     dp[i][j+1][mask] += val;
8e2e                 } else {
1e68                     if (d1 == d2 and not (i == lasti and j + 1 == lastj))
b333                         continue;
5ccf                     mask = Unite(Set(mask, j, 0, 0), d1, d2);
a611                     dp[i][j+1][mask] += val;
95cf                 }
8e2e             } else {
9e0c                 if (d1 == 0 and d2 == 0) dp[i][j+1][mask] += val;
95cf             }
95cf         }
95cf     }
faf8     cout << dp[lasti][lastj][0] << endl;
7021     return 0;
95cf }
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
const int SIZE = 10005;    2836
    427e
struct kmp_matcher {    d02b
  char p[SIZE];    2d81
  int fail[SIZE];    9847
  int len;    57b7
    427e
  void construct(const char* needle) {    60cf
    len = strlen(p);    aaa1
    strcpy(p, needle);    3a87
    fail[0] = fail[1] = 0;    3dd4
    for (int i = 1; i < len; i++) {    d8a8
      int j = fail[i];    147f
      while (j && p[i] != p[j]) j = fail[j];    3c79
      fail[i + 1] = p[i] == p[j] ? j + 1 : 0;    4643
    }    95cf
  }    95cf
    427e
  inline void found(int pos) {    c464
    // ! add codes for having found at pos    427e
  }    95cf
    427e
  void match(const char* haystack) {  // must be called after construct    2daf
    const char* t = haystack;    700f
    int n = strlen(t);    8482
    int j = 0;    8fd0
    rep(i, n) {    be8e
      while (j && p[j] != t[i]) j = fail[j];    4e19
      if (p[j] == t[i]) j++;    b5d5
      if (j == len) found(i - len + 1);    f024
    }    95cf
  }    95cf
};    329b
```

3.2 Manacher algorithm

```
struct Manacher {    81d4
  int Len;    cd09
```

```
9255   vector<int> lc;
b301   string s;
427e
ec07   void work() {
c033     lc[1] = 1;
6bef     int k = 1;
427e
491f     for (int i = 2; i <= Len; i++) {
7957       int p = k + lc[k] - 1;
5e04       if (i <= p) {
24a1         lc[i] = min(lc[2 * k - i], p - i + 1);
8e2e       } else {
e0e5         lc[i] = 1;
95cf       }
74ff       while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
2b9a       if (i + lc[i] > k + lc[k]) k = i;
95cf     }
95cf   }
427e
bfd5   void init(const char *tt) {
aaaf     int len = strlen(tt);
f701     s.resize(len * 2 + 10);
7045     lc.resize(len * 2 + 10);
8e13     s[0] = '*';
ae54     s[1] = '#';
1321     for (int i = 0; i < len; i++) {
e995       s[i * 2 + 2] = tt[i];
69fd       s[i * 2 + 1] = '#';
95cf     }
43fd     s[len * 2 + 1] = '#';
75d1     s[len * 2 + 2] = '\0';
61f7     Len = len * 2 + 2;
3e7a     work();
95cf   }
427e
b194   pair<int, int> maxpal(int l, int r) {
901a     int center = l + r + 1;
ffb2     int rad = lc[center] / 2;
ab54     int rmid = (l + r + 1) / 2;
17e4     int rl = rmid - rad, rr = rmid + rad - 1;
3908     if ((r ^ l) & 1) {
69f3     } else rr++;
69dc     return {max(l, rl), min(r, rr)};
95cf   }
```

```
};                                                        329b
```

3.3 Aho-corasick automaton

```
struct AC : Trie {                                        a1ad
  int fail[MAXN];                                         9143
  int last[MAXN];                                         daca
                                                          427e
  void construct() {                                      8690
    queue<int> q;                                         93d2
    fail[0] = 0;                                          a7a6
    rep(c, CHARN) {                                       ce3c
      if (int u = tr[0][c]) {                             b1c6
        fail[u] = 0;                                      a506
        q.push(u);                                        3e14
        last[u] = 0;                                      f689
      }                                                   95cf
    }                                                     95cf
    while (!q.empty()) {                                  cc78
      int r = q.front();                                  31f0
      q.pop();                                            15dd
      rep(c, CHARN) {                                     ce3c
        int u = tr[r][c];                                 ab59
        if (!u) {                                         0ef5
          tr[r][c] = tr[fail[r]][c];                      9d58
          continue;                                       b333
        }                                                 95cf
        q.push(u);                                        3e14
        int v = fail[r];                                  b3ff
        while (v && !tr[v][c]) v = fail[v];               d2ea
        fail[u] = tr[v][c];                               c275
        last[u] = tag[fail[u]] ? fail[u] : last[fail[u]]; 654c
      }                                                   95cf
    }                                                     95cf
  }                                                       95cf
                                                          427e
  void found(int pos, int j) {                            7752
    if (j) {                                              043e
      // ! add codes for having found word with tag[j]    427e
      found(pos, last[j]);                                4a96
    }                                                     95cf
  }                                                       95cf
```

```

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

3.4 Trie

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

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

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

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

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

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

3.5 Suffix array

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

Usage:

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

```
void radix_sort(int x[], int y[], int sa[], int n, int m) {
    static int cnt[1000005];    // size > max(n, m)
    fill(cnt, cnt + m, 0);
    rep (i, n) cnt[x[y[i]]]++;
    partial_sum(cnt, cnt + m, cnt);
    for (int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
}

void suffix_array(int s[], int sa[], int rk[], int n, int m) {
    static int y[1000005];  // size > n
    copy(s, s + n, rk);
    iota(y, y + n, 0);
```

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

3.6 Rolling hash

PLEASE call `init_hash()` in `int main()`!

Usage:

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

```
1e42  const LL mod = 1006658951440146419, g = 967;
9f60  const int MAXN = 200005;
0291  LL pg[MAXN];
427e
dfe7  inline LL mul(LL x, LL y) { return __int128_t(x) * y % mod; }
427e
599a  void init_hash() {   // must be called in `int main()`
286f      pg[0] = 1;
4af8      for (int i = 1; i < MAXN; i++) pg[i] = mul(pg[i-1], g);
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

Solve $ax + by = g = \gcd(a, b)$ w.r.t. x, y.
If (x_0, y_0) is an integer solution of $ax + by = g = \gcd(x, y)$, then every integer solution of it can be written as $(x_0 + kb', y_0 - ka')$, where $a' = a/g$, $b' = b/g$, and k is arbitrary integer.

```
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 Linear basis

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

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

4.3 Gauss elimination over finite field

```
const LL p = 1000000007;

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

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

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

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

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

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

4.4 Berlekamp-Massey algorithm

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

```
LL mod = 1000000007;
vector<LL> berlekamp(const vector<LL>& a) {
    vector<LL> p = {1}, r = {1};
    LL dif = 1;
    rep (i, a.size()) {
        LL u = 0;
        rep (j, p.size()) u = (u + p[j] * a[i-j]) % mod;
        if (u == 0) {
            r.insert(r.begin(), 0);
        } else {
            auto op = p;
            p.resize(max(p.size(), r.size() + 1));
            LL idif = powmod(dif, mod - 2);
            rep (j, r.size())
                p[j+1] = (p[j+1] - r[j] * idif % mod * u % mod + mod) % mod;
            dif = u; r = op;
        }
    }
    return p;
}
```

4.5 Fast Walsh-Hadamard transform

```
void fwt(int* a, int n){
    for (int d = 1; d < n; d <<= 1)
        for (int i = 0; i < n; i += d << 1)
            rep (j, d){
                int x = a[i+j], y = a[i+j+d];
                // a[i+j] = x+y, a[i+j+d] = x-y;    // xor
                // a[i+j] = x+y;                    // and
                // a[i+j+d] = x+y;                  // or
            }
}

void ifwt(int* a, int n){
    for (int d = 1; d < n; d <<= 1)
        for (int i = 0; i < n; i += d << 1)
            rep (j, d){
                int x = a[i+j], y = a[i+j+d];
                // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor
                // a[i+j] = x-y;                            // and
                // a[i+j+d] = y-x;                          // or
            }
}

void conv(int* a, int* b, int n){
    fwt(a, n);
    fwt(b, n);
    rep(i, n) a[i] *= b[i];
    ifwt(a, n);
}
```

4.6 Fast fourier transform

```
const int NMAX = 1<<20;

typedef complex<double> cplx;

const double PI = 2*acos(0.0);
struct FFT{
    int rev[NMAX];
    cplx omega[NMAX], oinv[NMAX];
    int K, N;

    FFT(int k){
        K = k; N = 1 << k;
        rep (i, N){
            rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
            omega[i] = polar(1.0, 2.0 * PI / N * i);
            oinv[i] = conj(omega[i]);
        }
    }

    void dft(cplx* a, cplx* w){
        rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int l = 2; l <= N; l *= 2){
            int m = l/2;
            for (cplx* p = a; p != a + N; p += l)
                rep (k, m){
                    cplx t = w[N/l*k] * p[k+m];
                    p[k+m] = p[k] - t; p[k] += t;
                }
        }
    }
```

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

4.7 Number theoretic transform

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

4.8 Sieve of Euler

```
const int MAXX = 1e7+5;        cfc3
bool p[MAXX];                  5861
int prime[MAXX], sz;           73ae
                               427e
```

```
void sieve(){
    p[0] = p[1] = 1;
    for (int i = 2; i < MAXX; i++){
        if (!p[i]) prime[sz++] = i;
        for (int j = 0; j < sz && i*prime[j] < MAXX; j++){
            p[i*prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
}
```

4.9 Sieve of Euler (General)

```
namespace sieve {
  constexpr int MAXN = 10000007;
  bool p[MAXN]; // true if not prime
  int prime[MAXN], sz;
  int pval[MAXN], pcnt[MAXN];
  int f[MAXN];

  void exec(int N = MAXN) {
    p[0] = p[1] = 1;

    pval[1] = 1;
    pcnt[1] = 0;
    f[1] = 1;

    for (int i = 2; i < N; i++) {
      if (!p[i]) {
        prime[sz++] = i;
        for (LL j = i; j < N; j *= i) {
          int b = j / i;
          pval[j] = i * pval[b];
          pcnt[j] = pcnt[b] + 1;
          f[j] = ______; // f[j] = f(i^pcnt[j])
        }
      }
      for (int j = 0; i * prime[j] < N; j++) {
        int x = i * prime[j]; p[x] = 1;
        if (i % prime[j] == 0) {
          pval[x] = pval[i] * prime[j];
          pcnt[x] = pcnt[i] + 1;
        } else {
          pval[x] = prime[j];
          pcnt[x] = 1;
        }
        if (x != pval[x]) {
          f[x] = f[x / pval[x]] * f[pval[x]]
        }
        if (i % prime[j] == 0) break;
      }
    }
  }
}
```

4.10 Miller-Rabin primality test

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

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

```
bool test(LL n){
    if (n < 3) return n==2;
    // ! The array a[] should be modified if the range of x changes.
    const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
    LL r = 0, d = n-1, x;
    while (~d & 1) d >>= 1, r++;
    for (int i=0; a[i] < n; i++){
        x = powmod(a[i], d, n); // ! powmod must use for 64bit mulmod
        if (x == 1 || x == n-1) goto next;
        rep (i, r) {
            x = mulmod(x, x, n);
            if (x == n-1) goto next;
        }
        return false;
next:;
    }
    return true;
}
```

4.11 Integer factorization (Pollard's rho)

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

4.12 Adaptive Simpson's Method

The Simpson's formula has order 3 algebraic precision.

Usage:

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

```
b7ec template <typename T>
9c6c double simpson(double l, double r, T&& f) {
38f4     double mid = (l + r) / 2;
2075     return (f(l) + 4 * f(mid) + f(r)) * (r - l) / 6.0;
95cf }
427e
b7ec template <typename T>
9cbb double integrate(double l, double r, double eps, double est, T&& f) {
38f4     double mid = (l + r) / 2;
5d09     double lv = simpson(l, mid, f), rv = simpson(mid, r, f);
d589     if (fabs(lv + rv - est) <= 15.0 * eps)
036c         return lv + rv + (lv + rv - est) / 15.0;
13c4     return integrate(l, mid, eps, lv, f) + integrate(mid, r, eps, rv, f);
95cf }
```

4.13 Linear Programming (Simplex)

This function solves the following linear program

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

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

```
typedef vector<double> VD;                                              db00
typedef vector<VD> VVD;                                                 9952
typedef vector<int> VI;                                                 89a3
const double EPS = 1e-9;                                                05b7
                                                                        427e
double LPSolve(VVD A, VD b, VD c, VD& x) {                              5eb7
    int m = b.size(), n = c.size();                                     f1f6
    VI B(m), N(n+1);                                                    1684
    VVD D(m+2, VD(n+2));                                                319d
    rep (i, m) rep (j, n) D[i][j] = A[i][j];                            7f8f
    rep (i, m) { B[i] = n + i; D[i][n] = -1; D[i][n+1] = b[i]; }        6b6c
    rep (j, n) { N[j] = j; D[m][j] = -c[j]; }                           9166
    N[n] = -1; D[m+1][n] = 1;                                           0def
                                                                        427e
    auto pivot = [&] (int r, int s) {                                   e0f7
        double inv = 1.0 / D[r][s];                                     3c4b
        rep (i, m+2) if (i != r) rep (j, n+2) if (j != s)               e090
            D[i][j] -= D[r][j] * D[i][s] * inv;                         48ea
        rep (j, n+2) if (j != s) D[r][j] *= inv;                        79f3
        rep (i, m+2) if (i != r) D[i][s] *= -inv;                       73cf
        D[r][s] = inv; swap(B[r], N[s]);                                82f1
    };                                                                  329b
                                                                        427e
    auto simplex = [&](int phase) {                                     3f89
        int x = m + (phase == 1);                                       adb8
        while (true) {                                                  1026
            int s = -1;                                                 0676
            for (int j = 0; j <= n; j++) {                              7e4d
                if (phase == 2 and N[j] == -1) continue;                30f5
                if (s == -1 or D[x][j] < D[x][s] or                     537c
                    D[x][j] == D[x][s] and N[j] < N[s]) s = j;          3262
            }                                                           95cf
```

```
            if (s < 0 or D[x][s] > -EPS) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < EPS) continue;
                if (r == -1 or D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] or
                    D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] and
                    B[i] < B[r]) r = i;
            }
            if (r == -1) return false; else pivot(r, s);
        }
    };

    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] <= -EPS) {
        pivot(r, n);
        if (!simplex(1) or D[m+1][n+1] < -EPS) return NAN;
        rep (i, m) if (B[i] == -1) {
            int s = -1;
            for (int j = 0; j <= n; j++) if (s == -1 or D[i][j] < D[i][s]
                or D[i][j] == D[i][s] and N[j] < N[s]) s = j;
            pivot(i, s);
        }
    }
    if (!simplex(2)) return DBL_MAX;
    x = VD(n);
    rep (i, m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return D[m][n+1];
}
```

5 Graph Theory

5.1 Strongly connected components

Usage:

`dfs(u)`	Run `dfs(u)` for each unlabelled vertex.
`scc[i]`	The vertices of the i-th scc.
`sccid[u]`	The index of the scc that contains u.
`contract()`	Compute the contracted graph.

```
const int MAXN = 100005;
int n, m;
vector<int> adj[MAXN];
int dfn[MAXN], low[MAXN], idx;
int sccid[MAXN], sccn;
vector<int> scc[MAXN];

void dfs(int u) {
    static stack<int> s;
    dfn[u] = low[u] = ++idx;
    s.push(u);
    for (int v : adj[u]) {
        if (!dfn[v]) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        } else if (!sccid[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        sccn++;
        do {
            sccid[s.top()] = sccn;
            scc[sccn].push_back(s.top());
            s.pop();
        } while (scc[sccn].back() != u);
    }
}

vector<int> adjc[MAXN];
void contract() {
    Rep (u, n) for (int v : adj[u]) if (sccid[u] != sccid[v])
        adjc[sccid[u]].push_back(sccid[v]);
}
```

5.2 Vertex biconnected components, cut vertex

A component root u is a cut vertex iff the size of `bccin[u]` is at least 2; for any other vertice u, it is a cut vertex iff `bccin[u]` is nonempty.

Usage:

`dfs(u)`	Run `dfs(u)` for each connected component.
`bcc[i]`	The edges of the i-th biconnected components, numbered from 0. If the bcc is a simple cycle, the edges are sorted in order.
`bccin[u]`	The indices of biconnected components reachable from vertex u.

```
0f42 const int MAXN = 100005;
35b8 int n, m;
0b32 vector<int> adj[MAXN];
0a8f int dfn[MAXN], low[MAXN], idx = 0;
05d2 vector<int> bccin[MAXN];
2eab vector<vector<pair<int, int>>> bcc;
3eed stack<pair<int, int>> st;
427e
6576 void dfs(int u, int p = 0) {
9891     dfn[u] = low[u] = ++idx;
18f6     for (int v : adj[u]) {
3c64         if (!dfn[v]) {
c600             st.emplace(u, v);
e2f7             dfs(v, u);
a19f             low[u] = min(low[u], low[v]);
9cb7             if (low[v] >= dfn[u]) {
a0e8                 bccin[u].push_back(bcc.size());
7dc7                 vector<pair<int, int>> cur;
a69f                 do {
bfe3                     cur.push_back(st.top());
b439                     st.pop();
5f33                 } while (cur.back() != make_pair(u, v));
b854                 reverse(range(cur));
0c6c                 bcc.push_back(move(cur));
95cf             }
dddc         } else if (dfn[v] < dfn[u] and v != p) {
c600             st.emplace(u, v);
769a             low[u] = min(low[u], dfn[v]);
95cf         }
95cf     }
95cf }
```

5.3 Minimum spanning arborescence, faster

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

Usage:

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

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

```
const int MAXN = 300005;                                                         5ece
typedef pair<LL, int> pii;                                                       2fef
struct MDST {                                                                    1495
    priority_queue<pii, vector<pii>, greater<pii>> heap[MAXN];                   01b2
    LL shift[MAXN];                                                              321d
    int fa[MAXN], vis[MAXN];                                                     fc06
                                                                                 427e
    int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }             38dd
                                                                                 427e
    void unite(int x, int y) {                                                   29b0
        x = find(x); y = find(y); fa[y] = x; if (x == y) return;                 0c14
        if (heap[x].size() < heap[y].size()) {                                   6fa0
            swap(heap[x], heap[y]);                                              9c26
            swap(shift[x], shift[y]);                                            2ffc
        }                                                                        95cf
        while (heap[y].size()) {                                                 9959
            auto p = heap[y].top(); heap[y].pop();                               175b
            heap[x].emplace(p.first - shift[y] + shift[x], p.second);            c0c5
        }                                                                        95cf
    }                                                                            95cf
                                                                                 427e
    void add_edge(int u, int v, LL w) { heap[v].emplace(w, u); }                 0bbd
                                                                                 427e
    LL run(int n, int rt) {                                                      a526
        LL ans = 0;                                                              f7ff
        iota(fa, fa + n + 1, 0);                                                 81f2
        Rep (i, n) if (find(i) != find(rt)) {                                    19b3
            int u = find(i);                                                     a7b1
            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];                       b385
                    shift[u] = heap[u].top().first;                              dde2
                    if (find(heap[u].top().second) != u) break;                  da47
                    heap[u].pop();                                               9fbb
                }                                                                95cf
```

```
                if (heap[u].empty()) return LLONG_MIN;
                u = find(heap[u].top().second);
            }
            while (s.size()) { vis[s.top()] = 0; unite(rt, s.top()); s.pop(); }
        }
        return ans;
    }
};
```

5.4 Minimum spanning arborescence, slow

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

Usage:

`init(n)`	Initalize the structure with n vertices, indexed from 1.
`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: $O(|V|^2)$

```
struct MDST {
    int V;
    LL heap[MAXN][MAXN];
    LL shift[MAXN];
    int fa[MAXN], vis[MAXN];

    void init(int n) {
        V = n;
        Rep (i, n) Rep (j, n) heap[i][j] = LLONG_MAX / 2;
    }

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

    void unite(int x, int y) {
        x = find(x); y = find(y); fa[y] = x; if (x == y) return ;
        Rep (i, V) heap[x][i] = min(heap[x][i], heap[y][i] - shift[y] + shift[x]);
    }

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

    LL run(int n, int rt) {
        V = n;
        LL ans = 0;
        iota(fa, fa + n + 1, 0);
        Rep (i, n) if (find(i) != find(rt)) {
            int u = find(i);
            stack<int, vector<int>> s;
            while (find(u) != find(rt)) {
                if (vis[u]) while (s.top() != u) {
                    vis[s.top()] = 0; unite(u, s.top()); s.pop();
                } else { vis[u] = 1; s.push(u); }

                Rep (i, V) if (find(i) == u) heap[u][i] = LLONG_MAX / 2;

                auto ptr = min_element(heap[u] + 1, heap[u] + V + 1);
                if (*ptr == LLONG_MAX / 2) return LLONG_MIN;
                ans += *ptr - shift[u];
                shift[u] = *ptr;

                u = ptr - heap[u];
            }
            while (s.size()) { vis[s.top()] = 0; unite(rt, s.top()); s.pop(); }
        }
        return ans;
    }
};
```

5.5 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{
    int from, to;
    LL cap, flow;
};

const int MAXN = 1005;
struct Dinic {
    int n, m, s, t;
    vector<edge> edges;
    vector<int> G[MAXN];
```

```
bbb6    bool vis[MAXN];
b40a    int d[MAXN];
ddec    int cur[MAXN];
427e
5973    void add_edge(int from, int to, LL cap) {
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
        return flow;                                                               84fb
    }                                                                              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
```

5.6 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;
        this->ny = ny;
        mx.resize(nx); my.resize(ny);
        e.clear(); e.resize(nx);
        mark.resize(nx);
    }

    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;
    }
};
```

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

Usage:

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

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

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

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

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

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

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

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

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

5.8 Minimum cost maximum flow

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

const LL INF = LLONG_MAX / 2;
const int MAXN = 5005;
struct MCMF {
    int s, t, n, m;
    vector<edge> edges;
    vector<int> G[MAXN];
    bool inq[MAXN]; // queue
    LL d[MAXN];     // distance
    int p[MAXN];    // previous
    int a[MAXN];    // improvement

    void add_edge(int from, int to, int cap, LL cost) {
        edges.push_back(edge{from, to, cap, 0, cost});
        edges.push_back(edge{to, from, 0, 0, -cost});
        m = edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }

    bool spfa(){
        queue<int> q;
        fill(d, d + MAXN, INF); d[s] = 0;
        memset(inq, 0, sizeof(inq));
        q.push(s); inq[s] = true;
        p[s] = 0; a[s] = INT_MAX;
        while (!q.empty()){
            int u = q.front(); q.pop(); inq[u] = false;
            for (int i : G[u]) {
                edge& e = edges[i];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = i;
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
                }
            }
        }
        return d[t] != INF;
    }

    void augment(){
        int u = t;
```

```
        while (u != s){
            edges[p[u]].flow += a[t];
            edges[p[u]^1].flow -= a[t];
            u = edges[p[u]].from;
        }
    }

#ifdef GIVEN_FLOW
    bool min_cost(int s, int t, int f, LL& cost) {
        this->s = s; this->t = t;
        int flow = 0;
        cost = 0;
        while (spfa()) {
            augment();
            if (flow + a[t] >= f){
                cost += (f - flow) * d[t]; flow = f;
                return true;
            } else {
                flow += a[t]; cost += a[t] * d[t];
            }
        }
        return false;
    }
#else
    int min_cost(int s, int t, LL& cost) {
        this->s = s; this->t = t;
        int flow = 0;
        cost = 0;
        while (spfa()) {
            augment();
            flow += a[t]; cost += a[t] * d[t];
        }
        return flow;
    }
#endif
};
```

5.9 Fast LCA, Virtual Tree

All indices of the tree are 1-based.

Usage:

`prep()`	Initialization.
`lca(u, v)`	Query the lowest common ancestor of u and v.
`vtree(vs)`	Create virtual tree with vertex set `vs`.

```
const int MAXN = 100005, root = 1;
int n;
vector<int> adj[MAXN];
int fa[MAXN], dfn[MAXN], dep[MAXN], idx;
pair<int, int> st[MAXN * 2][33 - __builtin_clz(MAXN)];

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

void dfs(int u, int p, int d) {
    fa[u] = p; dep[u] = d;
    st[dfn[u] = idx++][0] = {d, u};
    for (int v : adj[u]) if (v != p) {
        dfs(v, u, d + 1);
        st[idx++][0] = {d, u};
    }
}

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

vector<int> vadj[MAXN];
bool in[MAXN]; // is original vertex

struct vtree {
    vector<int> cvs;

    vtree(vector<int> vs) {
        for (int x : vs) in[x] = true;
        vs.push_back(root); // add root for convenience
        sort(range(vs), [] (int u, int v) { return dfn[u] < dfn[v]; });
        vs.erase(unique(range(vs)), vs.end());
        cvs = vs;
```

```
        vector<int> s;
        for (int x : vs) {
            if (s.empty()) {
                s.push_back(x);
            } else {
                int z = lca(x, s.back());
                while (s.size() > 1 and dep[z] < dep[s.rbegin()[1]]) {
                    int v = s.back(); s.pop_back();
                    vadj[s.back()].push_back(v);
                }
                if (dep[z] < dep[s.back()]) {
                    vadj[z].push_back(s.back());
                    s.pop_back();
                }
                if (s.empty() or s.back() != z) {
                    s.push_back(z);
                    cvs.push_back(z);
                }
                s.push_back(x);
            }
        }
        while (s.size() > 1) {
            int v = s.back(); s.pop_back();
            vadj[s.back()].push_back(v);
        }
    }

    int work(); // solve the subproblem

    ~vtree() {
        for (int x : cvs) {
            in[x] = false; vadj[x].clear();
            // do extra cleanup here
        }
    }

};
```

5.10 Heavy-light decomposition

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

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

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

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

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

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

```
6083     if (depth[u] > depth[v]) swap(u, v);
427e     // id[u] to id[v]
95cf }
```

5.11 Centroid decomposition

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

Usage:

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

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

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

```
  for (int v : adj[u]) { // divide and conquer    18f6
    adj[v].erase(find(range(adj[v]), u));        c375
    decompose(v);                                 fa6b
    adj[v].push_back(u); // restore deleted edge  a717
  }                                               95cf
}                                                 95cf
```

5.12 DSU on tree

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

Usage:

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

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

```
vector<int> adj[100005];                                                1fb6
int sz[100005], son[100005];                                            901d
                                                                        427e
void decomp(int u, int p) {                                             5559
    sz[u] = 1;                                                          50c0
    for (int v : adj[u]) {                                              18f6
        if (v == p) continue;                                           bd87
        decomp(v, u);                                                   a851
        sz[u] += sz[v];                                                 8449
        if (sz[v] > sz[son[u]]) son[u] = v;                             d28c
    }                                                                   95cf
}                                                                       95cf
                                                                        427e
template <typename T>                                                   b7ec
void trav(T fn, int u, int p) {                                         62f5
    fn(u);                                                              4412
    for (int v : adj[u]) if (v != p) trav(fn, v, u);                    30b3
}                                                                       95cf
                                                                        427e
#define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])   7467
void work(int u, int p, bool keep) {                                    33ff
    for_light(v) work(v, u, 0); // process light children               72a2
                                                                        427e
```

```
427e     // process heavy child
427e     // current data structure contains info of heavy child
9866     if (son[u]) work(son[u], u, 1);
427e
18a9     auto merge = [u] (int c) { /* count contribution of c */ };
1ab0     auto enter = [] (int c) { /* add vertex c */ };
f241     auto leave = [] (int c) { /* remove vertex c*/ };
427e
3d3b     for_light(v) {
74c6         trav(merge, v, u);
c13d         trav(enter, v, u);
95cf     }
427e
427e     // count answer for root and add it
427e     // 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
2d99     void init(int n) { tr.assign(N = n + 5, 0); }
427e
63d0     LL sum(int n) {
f7ff         LL ans = 0;
6770         while (n) { ans += tr[n]; n &= n - 1; }
4206         return ans;
95cf     }
427e
f4bd     void add(int n, LL x){
968e         while (n < N) { tr[n] += x; n += n & -n; }
95cf     }
329b };
```

6.2 Fenwick tree (range update point query)

```
struct bit_rupq{ // range update, point query                  3d03
    int N;                                                     d7af
    vector<LL> tr;                                             99ff
                                                               427e
    void init(int n) { tr.assign(N = n + 5, 0);}               2d99
                                                               427e
    LL query(int n) {                                          38d4
        LL ans = 0;                                            f7ff
        while (n < N) { ans += tr[n]; n += n & -n; }           3667
        return ans;                                            4206
    }                                                          95cf
                                                               427e
    void add(int n, LL x) {                                    f4bd
        while (n) { tr[n] += x; n &= n - 1; }                  0a2b
    }                                                          95cf
};                                                             329b
```

6.3 Segment tree

```
LL p;                                                          3942
const int MAXN = 4 * 100006;                                   1ebb
struct segtree {                                               451a
  int l[MAXN], m[MAXN], r[MAXN];                               27be
  LL val[MAXN], tadd[MAXN], tmul[MAXN];                        4510
                                                               427e
#define lson (o<<1)                                            ac35
#define rson (o<<1|1)                                          1294
                                                               427e
  void pull(int o) {                                           1344
    val[o] = (val[lson] + val[rson]) % p;                      bbe9
  }                                                            95cf
                                                               427e
  void push_add(int o, LL x) {                                 e4bc
    val[o] = (val[o] + x * (r[o] - l[o])) % p;                 5dd6
    tadd[o] = (tadd[o] + x) % p;                               6eff
  }                                                            95cf
                                                               427e
```

```cpp
  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 Mo's algorithm

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

Usage:

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

```cpp
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});
```

```cpp
}

int l, r;

// ----- functions to implement -----
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;
    for (int i = l; i <= r; i++) enter(i);
    for (query q : queries) {
        while (l > q.l) enter(--l);
        while (r < q.r) enter(++r);
        while (l < q.l) leave(l++);
        while (r > q.r) leave(r--);
        yield(q.id);
    }
}
```

6.5 Mo's algorithm on tree

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

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

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

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

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

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

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

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

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

6.6 Treap

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

Usage:

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

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

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

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

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

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

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

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

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

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

```
24b6 int root;
427e
d34f void init(int n) {
34d7     Rep (i, n) {
7681         int x; scanf("%d", &x);
0ed8         treap.Init(i, x);
bcc8         root = (i == 1) ? 1 : treap.Merge(root, i);
95cf     }
95cf }
427e
d030 void work(int op, int l, int r) {
6639     int tl, tm, tr;
b6c4     treap.Split(root, l, tl, tm);
8de3     treap.Split(tm, r - l, tm, tr);
3658     if (op == 1) {
c039         int x; scanf("%d", &x); treap.Add(tm, x);
1dcb     } else if (op == 2) {
ae78         treap.Reverse(tm);
581d     } else if (op == 3) {
e092         printf("%lld␣%d␣%d\n",
867f                 treap.sum[tm], treap.minv[tm], treap.maxv[tm]);
95cf     }
6188     root = treap.Merge(treap.Merge(tl, tm), tr);
95cf }
```

6.7 Link/cut tree

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

Usage:

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

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

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

```
      return y;
    }
};
```

6.8 Balanced binary search tree from `pb_ds`

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
  rkt;
// null_tree_node_update

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

6.9 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++);
}
```

6.10 Block list

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

Usage:

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

```
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)),
        minv(*min_element(range(data))),
        maxv(*max_element(range(data))),
        add(0), rev(0) { }

    void fix() {
        if (rev) reverse(range(data));          rev = 0;
        if (add) for (int& x : data) x += add;  add = 0;
    }

    void merge(block& another) {
        fix(); another.fix();
        vi temp(move(data));
        temp.insert(temp.end(), range(another.data));
        *this = block(move(temp));
    }

    block split(int pos) {
        fix();
        block result(vi(data.begin() + pos, data.end()));
        data.resize(pos); *this = block(move(data));
        return result;
    }
};

typedef list<block>::iterator lit;

struct blocklist {
    list<block> blk;

    void maintain() {
        lit it = blk.begin();
        while (it != blk.end() && next(it) != blk.end()) {
            lit it2 = it;
            while (next(it2) != blk.end() &&
                    it2->data.size() + next(it2)->data.size() <= BLOCK) {
                it2->merge(*next(it2));
                blk.erase(next(it2));
            }
            ++it;
        }
    }

    lit split(int pos) {
        for (lit it = blk.begin(); ; it++) {
            if (pos == 0) return it;
            while (it->data.size() > pos)
                blk.insert(next(it), it->split(pos));
            pos -= it->data.size();

        }
    }

    void Init(int *l, int *r) {
        for (int *cur = l; cur < r; cur += BLOCK)
            blk.emplace_back(vi(cur, min(cur + BLOCK, r)));
    }

    void Reverse(int l, int r) {
        lit it = split(l), it2 = split(r);
        reverse(it, it2);
        while (it != it2) {
            it->rev ^= 1;
            it++;
```

```
        }
        maintain();
    }

    void Add(int l, int r, int x) {
        lit it = split(l), it2 = split(r);
        while (it != it2) {
            it->sum += LL(x) * it->data.size();
            it->minv += x; it->maxv += x;
            it->add += x; it++;
        }
        maintain();
    }

    void Query(int l, int r) {
        lit it = split(l), it2 = split(r);
        LL sum = 0; int minv = INT_MAX, maxv = INT_MIN;
        while (it != it2) {
            sum += it->sum;
            minv = min(minv, it->minv);
            maxv = max(maxv, it->maxv);
            it++;
        }
        maintain();
        printf("%lld %d %d\n", sum, minv, maxv);
    }
} lst;
```

6.11 Persistent block list

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

When some operation that breaks block list property, please call `maintain` in time to restore the property.

Usage:

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

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

```
constexpr int BLOCK = 800;
typedef vector<int> vi;
typedef shared_ptr<vi> pvi;
typedef shared_ptr<const vi> pcvi;

struct block {
    pcvi data;
    LL sum;

    // add information to maintain
    block(pcvi ptr) :
        data(ptr),
        sum(accumulate(ptr->begin(), ptr->end(), 0ll))
    { }

    void merge(const block& another) {
        pvi temp = make_shared<vi>(data->begin(), data->end());
        temp->insert(temp->end(), another.data->begin(), another.data->end());
        *this = block(temp);
    }

    block split(int pos) {
        block result(make_shared<vi>(data->begin() + pos, data->end()));
        *this = block(make_shared<vi>(data->begin(), data->begin() + pos));
        return result;
    }
};

typedef list<block>::iterator lit;

struct blocklist {
    list<block> blk;

    void maintain() {
        lit it = blk.begin();
        while (it != blk.end() and next(it) != blk.end()) {
            lit it2 = it;
            while (next(it2) != blk.end() and
                    it2->data->size() + next(it2)->data->size() <= BLOCK) {
                it2->merge(*next(it2));
                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     }
329b };
```

6.12 Sparse table, range minimum query

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

```
db63 const int MAXN = 100007;
cefd int a[MAXN], st[MAXN][30];
427e
d34f void init(int n){
c73d     int l = log2(n);
cf75     rep (i, n) st[i][0] = a[i];
426b     rep (j, l) rep (i, 1+n-(1<<j))
1131         st[i][j+1] = min(st[i][j], st[i+(1<<j)][j]);
95cf }
427e
c863 int rmq(int l, int r){
f089     int k = log2(r - l);
    return min(st[l][k], st[r-(1<<k)][k]);    6117
}    95cf
```

7 Geometrics

7.1 2D geometric template

```
#include <bits/stdc++.h>    302f
using namespace std;    421c
    427e
typedef int T;    4553
typedef struct pt {    c0ae
    T x, y;    7a9d
    T operator , (pt a) { return x*a.x + y*a.y; } // inner product    ffaa
    T operator * (pt a) { return x*a.y - y*a.x; } // outer product    3ec7
    pt operator + (pt a) { return {x+a.x, y+a.y}; }    221a
    pt operator - (pt a) { return {x-a.x, y-a.y}; }    8b34
    427e
    pt operator * (T k) { return {x*k, y*k}; }    368b
    pt operator - () { return {-x, -y};}    90f4
} vec;    ba8c
    427e
typedef pair<pt, pt> seg;    0ea6
    427e
bool ptOnSeg(pt& p, seg& s){    8d6e
    vec v1 = s.first - p, v2 = s.second - p;    ce77
    return (v1, v2) <= 0 && v1 * v2 == 0;    de97
}    95cf
    427e
// 0 not on segment    427e
// 1 on segment except vertices    427e
// 2 on vertices    427e
int ptOnSeg2(pt& p, seg& s){    8421
    vec v1 = s.first - p, v2 = s.second - p;    ce77
    T ip = (v1, v2);    70ca
    if (v1 * v2 != 0 || ip > 0) return 0;    8b14
    return (v1, v2) ? 1 : 2;    0847
}    95cf
    427e
// if two orthogonal rectangles do not touch, return true    427e
inline bool nIntRectRect(seg a, seg b){    72bb
```

```
    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
// 4 collinear and have common point(s)
int intersect2(seg& a, seg& b){
    if (nIntRectRect(a, b)) return 0;
    vec va = a.second - a.first, vb = b.second - b.first;
    double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
           j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
    if (j1 < 0 || j2 < 0) return 0;
    if (j1 != 0 && j2 != 0) return 1;
    if (j1 == 0 && j2 == 0){
        if (va * vb == 0) return 4; else return 3;
    } else return 2;
}

template <typename Tp = T>
inline pt getIntersection(pt P, vec v, pt Q, vec w){
    static_assert(is_same<Tp, double>::value, "must␣be␣double!");
    return P + v * (w*(P-Q)/(v*w));
}

// -1 outside the polygon
// 0  on the border of the polygon
// 1  inside the polygon
int ptOnPoly(pt p, pt* poly, int n){
    int wn = 0;
    for (int i = 0; i < n; i++) {

        T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
        if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){
            if (k > 0 && d1 <= 0 && d2 > 0) wn++;
            if (k < 0 && d2 <= 0 && d1 > 0) wn--;
        } else return 0;
    }
    return wn ? 1 : -1;
}

istream& operator >> (istream& lhs, pt& rhs){
    lhs >> rhs.x >> rhs.y;
    return lhs;
}

istream& operator >> (istream& lhs, seg& rhs){
    lhs >> rhs.first >> rhs.second;
    return lhs;
}
```

8 Appendices

8.1 Number theory

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 Maximum number of divisors of n-digit number

d	max. #	first such number
1	4	6
2	12	60
3	32	840
4	64	7560
5	128	83160
6	240	720720
7	448	8648640
8	768	73513440
9	1344	735134400
10	2304	6983776800
11	4032	97772875200
12	6720	963761198400
13	10752	9316358251200
14	17280	97821761637600
15	26880	866421317361600
16	41472	8086598962041600
17	64512	74801040398884800
18	103680	897612484786617600

8.4 Burnside's lemma and Polya's enumeration theorem

8.4.1 Unweighted version

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.

8.4.2 Weighted version

For permutation $\pi \in G$, if π is a product of k cycles, and the ith cycle has length l_i, let

$$M_\pi(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} x_{l_i}.$$

The cycle index of G is defined by

$$P_G(x_1, x_2, \cdots, x_n) = \frac{1}{|G|}\sum_{\pi\in G} M_\pi(x_1, x_2, \cdots, x_n).$$

Given $\boldsymbol{v} = (n_1, n_2, \cdots, n_m)$ of nonnegative integers satisfying that $n_1+n_2+\cdots+n_m = n$, let $a_{\boldsymbol{v}}$ represent the number of nonequivalent m coloring of the n objects, where the ith color occurs precisely n_i times. The pattern inventory is the (multivariate) generating function for the sequence $a_{\boldsymbol{v}}$:

$$F_G(y_1, y_2, \cdots, y_m) = \sum_{\boldsymbol{v}} a_{\boldsymbol{v}} y_1^{n_1} y_2^{n_2} \cdots y_m^{n_m}$$

The weighted version of the Pólya's enumeration theorem says that

$$F_G(y_1, y_2, \cdots, y_m) = P_G(\sum_{i=1}^{m} y_i, \sum_{i=1}^{m} y_i^2, \cdots, \sum_{i=1}^{m} y_i^n)$$

8.5 Supnick TSP

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

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

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

8.6 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.

```
6dc9 #!/usr/bin/python2
4b2b from fractions import *
427e
796b def polymul(a, b) :
83e4     p = [0] * (len(a)+len(b)-1)
f697     for e1, c1 in enumerate(a) :
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)]
3cae print ' '.join(map(str, map(sum, zip(*map(
7c0d     lambda a, b : [x*a for x in b], y, lj)))))
```

8.7 LP duality

Primal	Dual
min z	max w
n variables	n constraints
var. ≥ 0	con. $\geq$
var. ≤ 0	con. $\leq$
free var.	con. $=$
m constraints	m variables
con. $\geq$	var. $\leq$
con. $\leq$	var. $\geq$
con. $=$	free var.
constraint vector	value vector
value vector	constraint vector

Primal:

$$\begin{aligned} \max \quad & z = 2x_1 + x_2 + 3x_3 + x_4 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 + x_4 \leq 5 \\ & 2x_1 - x_2 + 3x_3 = -4 \\ & x_1 - x_3 + x_4 \geq 1 \\ & x_1, x_3 \geq 0 \end{aligned}$$

Dual:

$$\begin{aligned} \min \quad & w = 5y_1 - 4y_2 + y_3 \\ \text{s.t.} \quad & y_1 + 2y_2 + y_3 \geq 2 \\ & y_1 - y_2 = 1 \\ & y_1 + 3y_2 - y_3 \geq 3 \\ & y_1 + y_3 = 1 \\ & y_1 \geq 0, y_3 \leq 0 \end{aligned}$$