TEAM NOTE

CRET

1. Mathematics 2
   1. ax + by = c 2
   2. number of relative prime number 2
   3. FFT, NTT with precision 2
   4. FFT, NTT cpp 5
   5. miller rabin, pollard rho 6
   6. sum of divisiors 7
   7. taylor series 8
2. Combinatorics 9
   1. lucas theorem 9
   2. stirling number 9
   3. catalan number 9
   4. Generating Function 9
3. Geometry 10
   1. line segment 10
   2. area of triangle 10
   3. convex hull 10
   4. Rotating Calipers 11
4. Graph 12
   1. bipartite matching 12
   2. scc 12
   3. scc – cpp 13
   4. Dinic 14
5. Tree 15
   1. HLD 15
6. String 18
   1. suffix array & LCP 18
   2. manacher 19
   3. kmp 20
7. Well-Known 21
   1. 2-sat 21
   2. second MST 21
   3. log LIS 23
   4. Aho-Corasik, trie 23
   5. Game-Theory 25
   6. Slope Trick 25
   7. DNC Optimization 25

Mathematics

ax + by = c

def **abcsol**(*a*, *b*, *c*):

    A = *a*

    B = *b*

    fa = [1, 0]

    fb = [0, 1]

*while* *a* % *b*:

        fa, fb = fb, [(x - y \* (*a* // *b*)) *for* x, y *in* zip(fa, fb)]

*a*, *b* = *b*, *a* % *b*

    gcd = *b*

*if* *c* % gcd:

*return* False

*return* [fb[i] \* *c* // gcd *for* i *in* range(2)]

print(abcsol(5, 2, 7))

*# (7, -14)*

number of relative prime number

*# phi using primes => []*

def **phi**(*n*, *primes*):

    rpn = *n*

*for* p *in* *primes*:

*if* *n* % p:

*continue*

*while* *n* % p == 0:

*n* //= p

        rpn //= p

        rpn \*= p - 1

*if* *n* == 1:

*break*

*if* *n* != 1:

        rpn \*= *n* - 1

        rpn //= *n*

*return* rpn

*# phi using factorization*

*from* math *import* \*

def **phi\_f**(*factor*):

*return* prod([fact - 1 *for* fact *in* list(set(*factor*))])

FFT, NTT with precision

*from* math *import* \*

def **FFT**(*f*, *w*):

    n = len(*f*)

*if* n == 1:

*return* *f*

    even = [*f*[i] *for* i *in* range(0, n, 2)]

    odd = [*f*[i] *for* i *in* range(1, n, 2)]

    even = FFT(even, *w* \*\* 2)

    odd = FFT(odd, *w* \*\* 2)

    wp = complex(1)

*for* i *in* range(n//2):

*f*[i] = even[i] + wp \* odd[i]

*f*[i + n//2] = even[i] - wp \* odd[i]

        wp \*= *w*

*return* *f*

*# A, B => index = degree*

def **multiple**(*A*, *B*):

    n = max(len(*A*), len(*B*))

    N = 2 \*\* ceil(log2(2 \* n))

*A* += [0] \* (N - len(*A*))

*B* += [0] \* (N - len(*B*))

    rw = complex(cos(tau / N), sin(tau / N))

*# FFT된 A와 B의 inner product*

    AA = FFT(*A*, rw)

    BB = FFT(*B*, rw)

    CC = [AA[i] \* BB[i] *for* i *in* range(N)]

*# inner product된 값을 다시 inverse FFT (1 / rw)*

    C = FFT(CC, complex(1) / rw)

*for* i *in* range(N):

        C[i] /= complex(N)

        C[i] = round(C[i].real)

*return* C

*from* math *import* \*

w = 3

mod1 = 2281701377

mod2 = 998244353

mod3 = 2130706433

def **power**(*a*, *b*, *mod*):

    ret = 1

*a* %= *mod*

*b* %= *mod*

*while* *b*:

*if* *b* & 1:

            ret = (ret \* *a*) % *mod*

*a* = (*a* \* *a*) % *mod*

*b* >>= 1

*return* ret

def **NTT**(*A*, *mod*, *inv*=False):

    n = len(*A*)

    rev = [0] \* n

*for* i *in* range(n):

        rev[i] = rev[i >> 1] >> 1

*if* i & 1:

            rev[i] |= n >> 1

*if* i < rev[i]:

*A*[i], *A*[rev[i]] = *A*[rev[i]], *A*[i]

    x = power(w, (*mod* - 1) // n, *mod*)

*if* *inv*:

        x = power(x, *mod* - 2, *mod*)

    root = [1]

*for* i *in* range(1, n):

        root.append((root[i-1] \* x) % *mod*)

    i = 2

*while* i <= n:

        step = n // i

*for* j *in* range(0, n, i):

*for* k *in* range(i>>1):

                u = *A*[j|k]

                v = (*A*[j|k|i >> 1] \* root[step\*k]) % *mod*

*A*[j|k] = (u + v) % *mod*

*A*[j|k|i >> 1] = (u - v) % *mod*

*if* *A*[j|k|i >> 1] < 0: *A*[j|k|i >> 1] += *mod*

        i <<= 1

*if* *inv*:

        t = power(n, *mod* - 2, *mod*)

*for* i *in* range(n):

*A*[i] = (*A*[i] \* t) % *mod*

*return* *A*

def **multiple**(*a*, *b*, *mod*):

    n = max(len(*a*), len(*b*))

    n = 2 \*\* ceil(log2(2 \* n))

*a* += [0] \* (n - len(*a*))

*b* += [0] \* (n - len(*b*))

    D = NTT(*a*, *mod*, *inv*=False)

    E = NTT(*b*, *mod*, *inv*=False)

*return* NTT([(D[i]\*E[i]) % *mod* *for* i *in* range(n)], *mod*, *inv*=True)

*import* copy

input()

a = list(map(int, input().split()))

b = list(map(int, input().split()))

c = copy.copy(a)

d = copy.copy(b)

C = multiple(c, d, mod1)

c = copy.copy(a)

d = copy.copy(b)

D = multiple(c, d, mod2)

E = multiple(a, b, mod3)

answer = []

*for* i *in* range(len(C)):

    ans = 0

    ans += C[i] \* mod2 \* mod3 \* power(mod2\*mod3, mod1-2, mod1)

    ans += D[i] \* mod1 \* mod3 \* power(mod1\*mod3, mod2-2, mod2)

    ans += E[i] \* mod1 \* mod2 \* power(mod1\*mod2, mod3-2, mod3)

    ans %= mod1 \* mod2 \* mod3

    answer.append(ans)

FFT, NTT cpp

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

#include <string>

#include <iomanip>

typedef long long ll;

using namespace std;

const ll mod = 2281701377;

const ll w = 3;

ll power(ll a, ll b) {

long long ret = 1;

while (b) {

if (b & 1)

ret = (ret \* a) % mod;

a = (1LL \* a \* a) % mod;

b /= 2;

}

return ret;

}

vector<ll> NTT(vector<ll>& A, bool inv=false) {

int n = A.size();

vector<ll> rev(n);

for (int i = 0; i < n; ++i) {

rev[i] = rev[i >> 1] >> 1;

if (i & 1)

rev[i] |= n >> 1;

if (i < rev[i])

swap(A[i], A[rev[i]]);

}

ll x = power(w, (mod - 1) / n);

if (inv) {

x = power(x, mod - 2);

}

vector<ll> root(n, 1);

for (int i = 1; i < n; ++i) {

root[i] = (root[i-1] \* x) % mod;

}

for (int i = 2; i <= n; i <<= 1) {

ll step = n / i;

for (int j = 0; j < n; j += i) {

for (int k = 0; k < (i >> 1); ++k) {

ll u = A[j|k];

ll v = (A[j|k|(i >> 1)] \* root[step\*k]) % mod;

A[j|k] = (u + v) % mod;

A[j|k|(i >> 1)] = (u - v) % mod;

if (A[j|k|(i >> 1)] < 0)

A[j|k|(i >> 1)] += mod;

}

}

}

if (inv) {

ll t = power(n, mod - 2);

for (int i = 0; i < n; ++i)

A[i] = (A[i] \* t) % mod;

}

return A;

}

vector<ll> multiply(vector<ll>& a, vector<ll>& b) {

int n = max(a.size(), b.size());

n = 2 \* pow(2, ceil(log2(n)));

a.resize(n);

b.resize(n);

vector<ll> A = NTT(a, false);

vector<ll> B = NTT(b, false);

vector<ll> result(n);

for (int i = 0; i < n; ++i){

result[i] = (A[i] \* B[i]) % mod;

}

return NTT(result, true);

}

miller rabin, polard rho

*import* math, random

def **power**(*x*, *y*, *p*):

    res = 1

    piv = *x* % *p*

*while* *y*:

*if* *y* & 1:

            res \*= piv

            res %= *p*

        piv \*= piv

        piv %= *p*

*y* >>= 1

*return* res

*# True => 합성수이다*

def **miller\_rabin**(*n*, *p*):

*if* *n* % *p* == 0:

*return* True

    d = *n* - 1

*while* 1:

        cur = power(*p*, d, *n*)

*if* cur == *n* - 1:

*return* False

*elif* d & 1:

*return* not (cur == 1 or cur == *n* - 1)

        d >>= 1

*# 소수면 True*

def **prime\_test**(*n*):

*for* i *in* [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]:

*if* *n* == i:

*return* True

*if* miller\_rabin(*n*, i):

*return* False

*return* True

def **func**(*c*, *x*, *n*):

*return* (*c* + ((*x* \*\* 2) % *n*)) % *n*

def **rho**(*n*, *factor*):

*if* *n* == 1:

*return* *factor* *if* *factor* *else* [1]

*if* *n* % 2 == 0:

*factor*.append(2)

*factor* = rho(*n* // 2, *factor*)

*return* *factor*

*if* prime\_test(*n*):

*factor*.append(*n*)

*return* *factor*

    a, b, c = 0, 0, 0

    g = *n*

*while* 1:

*if* g == *n*:

            b = random.randint(2, *n* - 1)

            a = b

            c = random.randint(1, 20)

        a = func(c, a, *n*)

        b = func(c, func(c, b, *n*), *n*)

        g = math.gcd(abs(a - b), *n*)

*if* g != 1:

*break*

*factor* = rho(g, *factor*)

*factor* = rho(*n* // g, *factor*)

*return* *factor*

sum of divisors

def **divsum**(*factor*):

    d = {}

*for* f *in* *factor*:

*if* f in d:

            d[f] += 1

*else*:

            d[f] = 1

    S = 1

*for* f, n *in* d.items():

        S \*= f \*\* (n+1) - 1

        S //= f-1

*return* S

taylor series

*from* decimal *import* Decimal

fact = [Decimal('0') *for* \_ *in* range(40)]

fact[1] = Decimal('1')

*for* i *in* range(2, 40):

    fact[i] = Decimal(str(i)) \* fact[i - 1]

def **sin**(*x*: Decimal):

*x* %= 2 \* Decimal('strinrg of phi')

    res = Decimal('0')

    value = *x*

*for* k *in* range(19):

        addition = (-1 *if* k % 2 == 1 *else* 1) \* (value / fact[2 \* k + 1])

        res += addition

        value \*= *x* \* *x*

*return* res

Combinatorics

lucas theorem

def **lucas**(*n*, *k*, *m*):

    ans = 1

*while* *k* != 0:

        ans = (ans \* math.comb(*n* % *m*, *k* % *m*)) % *m*

*n* //= *m*

*k* //= *m*

*return* ans

stirling number



폰트, 화이트, 친필, 서예이(가) 표시된 사진

자동 생성된 설명 텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명

catalan number

텍스트, 라인, 폰트, 친필이(가) 표시된 사진

자동 생성된 설명

Generating Function

텍스트, 친필, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

Geometry

line segment

A = [0, 0, 0, 0]

B = [0, 0, 0, 0]

def **CP**(*v1*, *v2*):

*return* *v1*[0] \* *v2*[1] - *v1*[1] \* *v2*[0]

def **PTV**(*p1*, *p2*):

*return* [*p2*[0] - *p1*[0], *p2*[1] - *p1*[1]]

def **intersection**(*A*, *B*):

*# True -> intersection*

*# False -> Not intersection*

    VA = PTV(*A*[:2], *A*[2:])

    AtoB1 = CP(VA, PTV(*A*[:2], *B*[:2]))

    AtoB2 = CP(VA, PTV(*A*[:2], *B*[2:]))

*if* AtoB1 \* AtoB2 > 0:

*return* False

    VB = PTV(*B*[:2], *B*[2:])

    BtoA1 = CP(VB, PTV(*B*[:2], *A*[:2]))

    BtoA2 = CP(VB, PTV(*B*[:2], *A*[2:]))

*if* BtoA1 \* BtoA2 > 0:

*return* False

*if* AtoB1 \* AtoB2 == 0 or BtoA1 \* BtoA2 == 0:

*if* max(*A*[0], *A*[2]) < min(*B*[0], *B*[2]):

*return* False

*if* max(*B*[0], *B*[2]) < min(*A*[0], *A*[2]):

*return* False

*if* max(*A*[1], *A*[3]) < min(*B*[1], *B*[3]):

*return* False

*if* max(*B*[1], *B*[3]) < min(*A*[1], *A*[3]):

*return* False

*return* True

area of triangle

def **area**(*p1*, *p2*, *p3*):

    A = *p1*[0] \* *p2*[1] + *p2*[0] \* *p1*[1] + *p3*[0] \* *p1*[1]

    B = *p1*[1] \* *p2*[0] + *p2*[1] \* *p1*[0] + *p3*[1] \* *p1*[0]

*return* abs(A - B)

convex hull

def **ccw**(*p1*, *p2*, *p3*):

*return* (*p2*[0] - *p1*[0]) \* (*p3*[1] - *p1*[1]) - (*p2*[1] - *p1*[1]) \* (*p3*[0] - *p1*[0])

def **convex\_hull**(*points*):

*points* = sorted(*points*)

    lower = []

*for* p *in* *points*:

*while* len(lower) >= 2 and ccw(lower[-2], lower[-1], p) <= 0:

            lower.pop()

        lower.append(p)

    upper = []

*for* p *in* reversed(*points*):

*while* len(upper) >= 2 and ccw(upper[-2], upper[-1], p) <= 0:

            upper.pop()

        upper.append(p)

*return* lower[:-1] + upper[:-1]

Rotating Calipers

def **cald**(*A*, *B*):

*return* (*A*[0] - *B*[0]) \*\* 2 + (*A*[1] - *B*[1]) \*\* 2

def **MD**(*t*):

    P = [(A[i][0] + *t*\*A[i][2], A[i][1] + *t*\*A[i][3]) *for* i *in* range(N)]

    HP = CH(P)

    M = 0

    L = len(HP)

    j = 0

*for* i *in* range(len(HP)):

        cv = (HP[i][0] - HP[(i+1)%L][0], HP[i][1] - HP[(i+1)%L][1])

*while* vccw((HP[(j+1)%L][0] - HP[j%L][0], HP[(j+1)%L][1] - HP[j%L][1]), cv) > 0:

            M = max(M, cald(HP[i], HP[j%L]))

            j += 1

        M = max(M, cald(HP[i], HP[j%L]), cald(HP[i], HP[(j+1)%L]))

*return* M

Graph

bipartite matching

L = []

R = []

E = [[] *for* \_ *in* range(len(L))]

S = [-1 *for* \_ *in* range((len(R)))]

V = [False *for* \_ *in* range(len(R))]

def **dfs**(*u*):

*for* v *in* E[*u*]:

*if* V[v]:

*continue*

        V[v] = True

*if* S[v] == -1 or dfs(S[v]):

            S[v] = *u*

*return* True

*return* False

*for* i *in* range(len(L)):

    V = [False *for* \_ *in* range(len(R))]

    dfs(i)

scc

*from* collections *import* deque

N = 1

E = [[] *for* \_ *in* range(N)]

F = [False *for* \_ *in* range(N)]

L = [0 *for* \_ *in* range(N)]

level = 0

ANS = []

S = deque([])

def **scc**(*u*):

    global level

    level += 1

    last = L[*u*] = level

    S.append(*u*)

*for* v *in* E[*u*]:

*if* not L[v]:

            last = min(last, scc(v))

*elif* not F[v]:

            last = min(last, L[v])

*if* last == L[*u*]:

        scc\_set = []

*while* S:

            p = S.pop()

            scc\_set.append(p)

            F[p] = True

*if* *u* == p:

*break*

        ANS.append(scc\_set)

*return* last

*for* i *in* range(N):

*if* not F[i]:

        scc(i)

SCC – CPP

class SCC{

public:

  int V;

  vector<vector<int>>& graph;

  int groupId = 0;

  vector<int> groupIdOf;

  vector<bool> visited;

  vector<int> stack;

  vector<int> stackIdx;

  const int MAX = 987654321;

  SCC(int *V*, vector<vector<int>>& *graph*):V(*V*), graph(graph){}

  vector<int> getScc(){

    groupIdOf = vector<int>(V, -1);

    stackIdx = vector<int>(V, -1);

    visited = vector<bool>(V, false);

    for (int v = 0; v < V; v++) {

      if (visited[v]) continue;

      DFS(v, stack, stackIdx);

    }

    return groupIdOf;

  }

  int DFS(int *curNode*, vector<int>& *stack*, vector<int>& *stackIdx*) {

    visited[*curNode*] = true;

*stack*.push\_back(*curNode*);

*stackIdx*[*curNode*] = *stack*.size();

    int minParentIdx = MAX;

    for (auto e : graph[*curNode*]) {

      if (*stackIdx*[e] != -1 && *stackIdx*[e] < minParentIdx) {

        minParentIdx = *stackIdx*[e];

        continue;

      }

      if (visited[e]) continue;

      minParentIdx = min(DFS(e, *stack*, *stackIdx*), minParentIdx);

    }

    if (minParentIdx == *stackIdx*[*curNode*] || minParentIdx == MAX) {

      while (*stack*.size() > 0 && *stack*.back() != *curNode*) {

        groupIdOf[*stack*.back()] = groupId;

*stackIdx*[*stack*.back()] = -1;

*stack*.pop\_back();

      }

      groupIdOf[*stack*.back()] = groupId;

*stackIdx*[*stack*.back()] = -1;

*stack*.pop\_back();

      groupId += 1;

      return MAX;

    }

    return minParentIdx;

  }

};

Dinic

*from* collections *import* deque

N = 1

F = [[0 *for* \_ *in* range(N)] *for* \_ *in* range(N)]

L = [-1 *for* \_ *in* range(N)]

P = [ 0 *for* \_ *in* range(N)]

ADJ = [[] *for* \_ *in* range(N)]

S, E = 0, N - 1

def **bfs**(*S*, *E*):

    global L

    L = [-1 *for* \_ *in* range(N)]

    L[*S*] = 0

    queue = deque([*S*])

*while* queue:

        u = queue.popleft()

*for* v *in* ADJ[u]:

*if* L[v] == -1 and F[u][v]:

                L[v] = L[u] + 1

                queue.append(v)

*return* L[*E*] != -1

def **dfs**(*u*, *f*, *E*):

    global P

*if* *u* == *E*:

*return* *f*

*while* P[*u*] <= *E*:

        v = P[*u*]

*if* L[*u*] < L[v] and F[*u*][v]:

            add = dfs(v, min(*f*, F[*u*][v]), *E*)

*if* add:

                F[*u*][v] -= add

                F[v][*u*] += add

*return* add

        P[*u*] += 1

*return* 0

MF = 0

*while* bfs(S, E):

    P = [0 *for* \_ *in* range(N + 2)]

*while* add := dfs(S, float('inf'), E):

        MF += add

Tree

HLD – Euler Route

vector<vector<int>> graph;

class hldNode{

public:

  int idx;

  int root;

  int leaf;

  int parent;

  int childEdIdx;

  int heavy;

  int level;

};

vector<hldNode> hldList;

vector<int> hldNodeQueue;

int **getChildCntAndInitHeavyNode**(int *parent*, int *idx*){

  int cnt = 0;

  int heavyIdx = -1;

  int heavyCnt = 0;

  for(auto e: graph[*idx*]){

    if(e == *parent*) continue;

    int curCnt = **getChildCntAndInitHeavyNode**(*idx*, e);

    cnt += curCnt;

    if(heavyCnt < curCnt){

      heavyCnt = curCnt;

      heavyIdx = e;

    }

  }

  hldList[*idx*].heavy = heavyIdx;

  return cnt + 1;

}

int **hldInit**(int *parent*, int *cur*, int *root*, int *level*){

  if(hldList[*cur*].heavy == -1){

    hldList[*cur*].idx = hldNodeQueue.size();

    hldList[*cur*].root = *root*;

    hldList[*cur*].leaf = *cur*;

    hldList[*cur*].parent = *parent*;

    hldList[*cur*].childEdIdx = hldList[*cur*].idx;

    hldList[*cur*].level = *level*;

    hldNodeQueue.push\_back(*cur*);

    return *cur*;

  }

  hldList[*cur*].idx = hldNodeQueue.size();

  hldList[*cur*].root = *root*;

  hldNodeQueue.push\_back(*cur*);

  hldList[*cur*].leaf = **hldInit**(*cur*, hldList[*cur*].heavy, *root*, *level* + 1);

  hldList[*cur*].parent = *parent*;

  for(auto e: graph[*cur*]){

    if(e == *parent* || e == hldList[*cur*].heavy) continue;

**hldInit**(*cur*, e, e, *level* + 1);

  }

  hldList[*cur*].childEdIdx = hldNodeQueue.size() - 1;

  hldList[*cur*].level = *level*;

  return hldList[*cur*].leaf;

}

int **lca**(int *a*, int *b*){

  while(*a* != *b*){

    if(hldList[*a*].root == hldList[*b*].root){

      if(hldList[*a*].level < hldList[*b*].level)

*b* = *a*;

      else

*a* = *b*;

    }else{

      if(hldList[hldList[*a*].root].level < hldList[hldList[*b*].root].level)

*b* = hldList[hldList[*b*].root].parent;

      else if(hldList[hldList[*a*].root].level > hldList[hldList[*b*].root].level)

*a* = hldList[hldList[*a*].root].parent;

      else{

*a* = hldList[*a*].parent;

*b* = hldList[*b*].parent;

      }

    }

  }

  return *a*;

}

class segNode{

public:

  int st;

  int ed;

  ll val;

  pair<ll, ll> lazy;

};

vector<segNode> segTree;

int leafNum = 1;

void **segInit**(int *N*){

  while(leafNum < *N*) leafNum \*= 2;

  segTree = vector<segNode>(leafNum \* 2);

  for(int i = 0; i < *N*; i++){

    segTree[leafNum + i].st = i;

    segTree[leafNum + i].ed = i + 1;

    segTree[leafNum + i].val = 0;

    segTree[leafNum + i].lazy = {1, 0};

  }

  for(int i = *N*; i < leafNum; i++){

    segTree[leafNum + i].st = i;

    segTree[leafNum + i].ed = i + 1;

    segTree[leafNum + i].val = 0;

    segTree[leafNum + i].lazy = {0, 0};

  }

  for(int i = leafNum - 1; i > 0; i--){

    segTree[i].st = segTree[2 \* i].st;

    segTree[i].ed = segTree[2 \* i + 1].ed;

    segTree[i].val = 0;

    segTree[i].lazy = {1, 0};

  }

}

void **segUpdate**(int *idx*, int *st*, int *ed*, pair<ll, ll> *op*){

  if(segTree[*idx*].st == *st* && segTree[*idx*].ed == *ed*){

    segTree[*idx*].lazy.first \*= *op*.first;

    segTree[*idx*].lazy.second \*= *op*.first;

    segTree[*idx*].lazy.second += *op*.second;

    segTree[*idx*].val = segTree[*idx*].val \* *op*.first + (*op*.second) \* (*ed* - *st*);

    segTree[*idx*].lazy.first %= DIV;

    segTree[*idx*].lazy.second %= DIV;

    segTree[*idx*].val %= DIV;

    return;

  }

  if(segTree[*idx*].lazy.first != 1 || segTree[*idx*].lazy.second != 0){

**segUpdate**(2 \* *idx*, segTree[2 \* *idx*].st, segTree[2 \* *idx*].ed, segTree[*idx*].lazy);

**segUpdate**(2 \* *idx* + 1, segTree[2 \* *idx* + 1].st, segTree[2 \* *idx* + 1].ed, segTree[*idx*].lazy);

    segTree[*idx*].lazy = {1, 0};

  }

  if(*ed* <= segTree[2 \* *idx*].ed)

**segUpdate**(2 \* *idx*, *st*, *ed*, *op*);

  else if(segTree[2 \* *idx* + 1].st <= *st*)

**segUpdate**(2 \* *idx* + 1, *st*, *ed*, *op*);

  else{

**segUpdate**(2 \* *idx*, *st*, segTree[2 \* *idx*].ed, *op*);

**segUpdate**(2 \* *idx* + 1, segTree[2 \* *idx* + 1].st, *ed*, *op*);

  }

  segTree[*idx*].val = segTree[2 \* *idx*].val + segTree[2 \* *idx* + 1].val;

  segTree[*idx*].val %= DIV;

  return;

}

ll **segGetVal**(int *idx*, int *st*, int *ed*){

  if(segTree[*idx*].st == *st* && segTree[*idx*].ed == *ed*){

    return segTree[*idx*].val;

  }

  if(segTree[*idx*].lazy.first != 1 || segTree[*idx*].lazy.second != 0){

**segUpdate**(2 \* *idx*, segTree[2 \* *idx*].st, segTree[2 \* *idx*].ed, segTree[*idx*].lazy);

**segUpdate**(2 \* *idx* + 1, segTree[2 \* *idx* + 1].st, segTree[2 \* *idx* + 1].ed, segTree[*idx*].lazy);

    segTree[*idx*].lazy = {1, 0};

    segTree[*idx*].val = segTree[2 \* *idx*].val + segTree[2 \* *idx* + 1].val;

    segTree[*idx*].val %= DIV;

  }

  if(*ed* <= segTree[2 \* *idx*].ed)

    return **segGetVal**(2 \* *idx*, *st*, *ed*);

  if(segTree[2 \* *idx* + 1].st <= *st*)

    return **segGetVal**(2 \* *idx* + 1, *st*, *ed*);

  return (**segGetVal**(2 \* *idx*, *st*, segTree[2 \* *idx*].ed) + **segGetVal**(2 \* *idx* + 1, segTree[2 \* *idx* + 1].st, *ed*)) % DIV;

}

String

suffix array & LCP

*import* math

def **LCPSUFFIX**(*S*, *L*):

    def **radix\_sort**(*rank*, *max\_rank*, *rktoi*, *L*):

        radix = [0 *for* \_ *in* range(*max\_rank* + 1)]

        new\_rktoi = [0 *for* \_ *in* range(*L*)]

*# radix를 cummulative로 구성*

*for* rk *in* range(*L*):

            radix[*rank*[*rktoi*[rk]]] += 1

*for* ra *in* range(*max\_rank*):

            radix[ra + 1] += radix[ra]

*# sorting 후의 ranking 위치로 i 값을 옮김*

*for* rk *in* range(*L* - 1, -1, -1):

            radix[*rank*[*rktoi*[rk]]] -= 1

            new\_rktoi[radix[*rank*[*rktoi*[rk]]]] = *rktoi*[rk]

*# rank로 sorting 된 rktoi반환*

*return* new\_rktoi

    def **update\_rank**(*rank1*, *rank2*, *rktoi*, *L*):

        rank\_count = 1

        new\_rank = [0 *for* \_ *in* range(*L*)]

        new\_rank[*rktoi*[0]] = 1

*for* i *in* range(1, *L*):

*if* *rank1*[*rktoi*[i - 1]] != *rank1*[*rktoi*[i]] or *rank2*[*rktoi*[i - 1]] != *rank2*[*rktoi*[i]]:

                rank\_count += 1

            new\_rank[*rktoi*[i]] = rank\_count

*# rank1, rank2를 기준으로 rank1의 동점자가 처리된 rank를 반환 및 rank 갯수 반환*

*return* new\_rank, rank\_count

    rank1 = [ord(*S*[i]) - ord('A') + 1 *for* i *in* range(*L*)]

    rank2 = [0 *for* \_ *in* range(*L*)]

    rktoi = radix\_sort(rank1, max(rank1), [i *for* i *in* range(*L*)], *L*)

    rank1, rank\_count = update\_rank(rank1, rank2, rktoi, *L*)

*for* i *in* range(math.ceil(math.log2(*L*))):

*for* rk *in* range(*L*):

            rank2[rktoi[rk]] = rank1[rktoi[rk] + 2 \*\* i] *if* rktoi[rk] + 2 \*\* i < *L* *else* 0

        rktoi = radix\_sort(rank2, rank\_count, rktoi, *L*)

        rktoi = radix\_sort(rank1, rank\_count, rktoi, *L*)

        rank1, rank\_count = update\_rank(rank1, rank2, rktoi, *L*)

*if* rank\_count == *L*:

*break*

    itork = [0 *for* \_ *in* range(*L*)]

*for* i *in* range(*L*):

        itork[rktoi[i]] = i

*# LCP배열 생성*

    LCP = [0 *for* \_ *in* range(*L*)]

    val = 0

*for* i *in* range(*L*):

*if* itork[i] == 0:

*continue*

        uprki = rktoi[itork[i] - 1]

*while* i + val < *L* and uprki + val < *L* and *S*[i + val] == *S*[uprki + val]:

            val += 1

        LCP[itork[i]] = val

        val = max(val - 1, 0)

*return* LCP

manacher

*import* sys

*# 필수 전역 변수*

S = sys.stdin.readline()[:-1]   *# 문자열*

S = "#".join(S)               *# 문자열 전처리*

N = len(S)                      *# 문자열 길이*

r = [0 *for* i *in* range(len(S))]  *# r[i] : i 번째 단어의 펠린드롬 반지름*

far = -1                          *# 가장 멀리 온 팰린드롬 끝부분*

farMid = -1                       *# 가장 멀리 온 팰린드롬 끝부분의 중심점*

*################################################################ 함수부*

*## ind의 펠린드롬 반지름 길이 구하는 함수*

def **getPalinRadius**(*ind*):

  global S, N, r, far, farMid

  curR = 0

*# curR 초기화 부분*

*if* *ind* <= far:

    curR = min(r[2 \* farMid - *ind*], far - *ind*)

*# 좌우로 넓혀가며 반지름 찾는 부분*

*while* 0 <= *ind* - curR - 1 and *ind* + curR + 1 < N and S[*ind* - curR - 1] == S[*ind* + curR + 1]:

    curR += 1

  r[*ind*] = curR

*# far 갱신*

*if* far < *ind* + curR:

    far = *ind* + curR

    farMid = *ind*

*return* r[*ind*]

*## 인덱스와 계산된 길이를 줬을 때 펠린드롬 길이를 구하는 함수*

*## getPalinRadius는 전처리된 반지름을 기준으로 하기 때문에 무작정 2 곱하면 안됨*

def **getPalinLen**(*ind*, *r*):

*if* *ind* % 2 == 0:  *# 실제 단어 일때*

*return* 1 + (*r* // 2) \* 2

*else*: *# 더미 단어 일때*

*return* (*r* + 1) // 2 \* 2

*################################################################ 실행부*

ans = 0

*for* i *in* range(len(S)):

  curAns = getPalinLen(i, getPalinRadius(i))

*if* ans < curAns:

    ans = curAns

print(ans)

kmp

int **main**() {

    string T;

    getline(cin, T);

    string P;

    getline(cin, P);

*//p배열 설정하기*

    vector<int> p(P.size(), 0);

    int curCompIdx = 0;

    for (int i = 1; i < P.size(); i++) {

        while (curCompIdx > 0 && P[curCompIdx] != P[i])

            curCompIdx = p[curCompIdx - 1];

        if (P[curCompIdx] == P[i])

            p[i] = ++curCompIdx;

    }

*//찾기*

    int curPatternIdx = 0;

    vector<int> idxArray;

    for (int k = 0; k < T.size(); k++) {

        if (curPatternIdx == P.size()) {

            idxArray.push\_back(k - P.size());

            curPatternIdx = p[curPatternIdx - 1];

        }

        while (curPatternIdx > 0 && P[curPatternIdx] != T[k])

            curPatternIdx = p[curPatternIdx - 1];

        if (P[curPatternIdx] == T[k])

            curPatternIdx++;

        else

            curPatternIdx = 0;

    }

    if (curPatternIdx == P.size()) {

        idxArray.push\_back(T.size() - P.size());

    }

    cout << idxArray.size() << "\n";

    for (auto e : idxArray)

        cout << e + 1 << " ";

    return 0;

}

Well-Known

2-sat

*for* i *in* range(N6):

*if* not finished[i]:

        scc(i)

res = [0] \* N3

*for* i *in* range(N3):

*if* scc\_num[i] == scc\_num[i + N3]:

        print(-1)

*break*

*if* scc\_num[i + N3] < scc\_num[i]:

        res[i] = 1

*else*:

    print(res.count(1))

*for* i *in* range(N3):

*if* res[i]:

            print(i + 1, *end*=' ')

second MST

두번째 MST

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

*//Graph*

int V, E;

vector<vector<int>> edges;

*//Union-Find*

vector<int> groupOf;

int **getGroup**(int *v*){

  if(*v* != groupOf[*v*]) groupOf[*v*] = **getGroup**(groupOf[*v*]);

  return groupOf[*v*];

}

void **unionGroup**(int *v1*, int *v2*){

*v1* = **getGroup**(*v1*);

*v2* = **getGroup**(*v2*);

  if(*v1* < *v2*)

    groupOf[*v2*] = *v1*;

  else

    groupOf[*v1*] = *v2*;

}

*//LCA*

vector<int> levelOf;

vector<vector<int>> spTable;

vector<vector<int>> spTableMax;

vector<vector<int>> spTable2ndMax;

void **initSpTable**(){

  for(int j = 1; j < 20; j++){

    for(int i = 0; i < V + 1; i++){

      spTable[i][j] = spTable[spTable[i][j - 1]][j - 1];

      spTableMax[i][j] = -1;

      spTable2ndMax[i][j] = -1;

      vector<int> curVals(4);

      curVals[0] = spTableMax[i][j - 1];

      curVals[1] = spTable2ndMax[i][j - 1];

      curVals[2] = spTableMax[spTable[i][j - 1]][j - 1];

      curVals[3] = spTable2ndMax[spTable[i][j - 1]][j - 1];

**sort**(curVals.rbegin(), curVals.rend());

      spTableMax[i][j] = curVals[0];

      for(int k = 1; k < 4; k++){

        if(curVals[k] == curVals[0]) continue;

        spTable2ndMax[i][j] = curVals[k];

        break;

      }

    }

  }

}

int **LCA**(int *v1*, int *v2*){

  int curMaxEdge = -1;

  while(levelOf[*v1*] < levelOf[*v2*]){

    curMaxEdge = **max**(curMaxEdge, spTableMax[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))]);

*v2* = spTable[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))];

  }

  while(levelOf[*v1*] > levelOf[*v2*]){

    curMaxEdge = **max**(curMaxEdge, spTableMax[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))]);

*v1* = spTable[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))];

  }

  while(*v1* != *v2*){

    int st = 1;

    int ed = 20;

    while(st <= ed){

      int mid = (st + ed) / 2;

      if(spTable[*v1*][mid] == spTable[*v2*][mid])

        ed = mid - 1;

      else

        st = mid + 1;

    }

    curMaxEdge = **max**(curMaxEdge, spTableMax[*v1*][ed]);

    curMaxEdge = **max**(curMaxEdge, spTableMax[*v2*][ed]);

*v1* = spTable[*v1*][ed];

*v2* = spTable[*v2*][ed];

  }

  return curMaxEdge;

}

int **LCA2nd**(int *v1*, int *v2*, int *maxEdge*){

  int cur2ndMaxEdge = -1;

  while(levelOf[*v1*] < levelOf[*v2*]){

    if(spTableMax[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTableMax[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))]);

    if(spTable2ndMax[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTable2ndMax[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))]);

*v2* = spTable[*v2*][(int)(**floor**(**log2**(levelOf[*v2*] - levelOf[*v1*])))];

  }

  while(levelOf[*v1*] > levelOf[*v2*]){

    if(spTableMax[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTableMax[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))]);

    if(spTable2ndMax[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTable2ndMax[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))]);

*v1* = spTable[*v1*][(int)(**floor**(**log2**(levelOf[*v1*] - levelOf[*v2*])))];

  }

  while(*v1* != *v2*){

    int st = 1;

    int ed = 20;

    while(st <= ed){

      int mid = (st + ed) / 2;

      if(spTable[*v1*][mid] == spTable[*v2*][mid])

        ed = mid - 1;

      else

        st = mid + 1;

    }

    if(spTableMax[*v1*][ed] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTableMax[*v1*][ed]);

    if(spTable2ndMax[*v1*][ed] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTable2ndMax[*v1*][ed]);

    if(spTableMax[*v2*][ed] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTableMax[*v2*][ed]);

    if(spTable2ndMax[*v2*][ed] < *maxEdge*)

      cur2ndMaxEdge = **max**(cur2ndMaxEdge, spTable2ndMax[*v2*][ed]);

*v1* = spTable[*v1*][ed];

*v2* = spTable[*v2*][ed];

  }

  return cur2ndMaxEdge;

}

*//MST*

int MSTWeight;

vector<bool> isMSTEdge;

vector<vector<pair<int, int>>> MST;

void **getMST**(){

**sort**(edges.begin(), edges.end());

  for(int i = 0; i < E; i++){

    if(**getGroup**(edges[i][1]) == **getGroup**(edges[i][2])) continue;

    MSTWeight += edges[i][0];

    isMSTEdge[edges[i][3]] = true;

**unionGroup**(edges[i][1], edges[i][2]);

    MST[edges[i][1]].push\_back({edges[i][2], edges[i][0]});

    MST[edges[i][2]].push\_back({edges[i][1], edges[i][0]});

  }

}

*//인접 리스트로 된 트리를 dfs를 통해 spTable 과 levelOf를 초기화 하기 위함*

void **dfs**(int *cur*, int *prv*, int *level*){

  levelOf[*cur*] = *level*;

  for(auto e: MST[*cur*]){

    if(e.first == *prv*) continue;

**dfs**(e.first, *cur*, *level* + 1);

    spTable[e.first][0] = *cur*;

    spTableMax[e.first][0] = e.second;

  }

}

log LIS

*import* sys

A = [1, 2, 3, 4, 5]

N = len(A)

D = [A[0]] *# 그 갯수인 최소값*

I = [0] *# 그 최소값의 index*

DA = [1 *for* \_ *in* range(N)] *# 모든 위치에서의 최대*

IA = [-1 *for* \_ *in* range(N)] *# 최대를 이루는 바로 이전 index*

def **find\_index**(*left*, *right*, *value*):

*while* *left* != *right*:

        mid = (*left* + *right*) >> 1

*if* D[mid] < *value*:

*left* = mid + 1

*elif* *value* <= D[mid]:

*right* = mid

*return* *left*

*for* i *in* range(1, N):

    index = -1

*if* A[i] > D[-1]:

        index = len(D)

        D.append(A[i])

        I.append(i)

*else*:

        index = find\_index(0, len(D) - 1, A[i])

*if* A[i] < D[index]:

            D[index] = A[i]

            I[index] = i

    DA[i] = index + 1

    IA[i] = -1 *if* index - 1 < 0 *else* I[index - 1]

ans = []

c = I[-1]

*while* c != -1:

    ans.append(A[c])

    c = IA[c]

print(ans)

Aho-Corasik, trie

#include <iostream>

#include <string>

#include <vector>

#include <queue>

using namespace std;

class Node {

public:

bool ifEnd;

vector<Node\*> children;

Node\* fail;

Node() {

ifEnd = false;

for (int i = 0; i < 26; i++)

children = vector<Node\*>(26, NULL);

fail = NULL;

}

};

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

cout.tie(NULL);

int N;

cin >> N;

//트라이 트리 구축

Node\* root = new Node();

for (int i = 0; i < N; i++) {

string str;

cin >> str;

Node\* curNode = root;

for (int j = 0; j < str.size(); j++) {

if (curNode->children[str[j] - 'a'] == NULL)

curNode->children[str[j] - 'a'] = new Node();

curNode = curNode->children[str[j] - 'a'];

}

curNode->ifEnd = true;

}

//실패함수 매핑

queue<Node\*> Queue;

//첫번째 레벨 node들의 fail은 루트로 지정해주고, Q에 넣기

for(int i = 0; i < 26; i++){

if(root->children[i] == NULL) continue;

root->children[i]->fail = root;

Queue.push(root->children[i]);

}

while (!Queue.empty()) {

Node\* curNode = Queue.front();

Queue.pop();

//현재 노드의 자식들의 실패함수를 매핑해주기.

for (int i = 0; i < 26; i++) {

if (curNode->children[i] == NULL) continue;

Node\* curFail = curNode->fail;

while (curFail != NULL && curFail->children[i] == NULL)

curFail = curFail->fail;

if (curFail != NULL){

if(curFail->children[i]->ifEnd)

curNode->children[i]->ifEnd = true;

curNode->children[i]->fail = curFail->children[i];

}

else{

curNode->children[i]->fail = root;

}

Queue.push(curNode->children[i]);

}

}

//탐색 수행

int Q;

cin >> Q;

for(int i = 0; i < Q; i++){

string str;

cin >> str;

Node\* curNode = root;

for(int j = 0; j < str.size(); j++){

if(curNode->children[str[j] - 'a'] != NULL){

curNode = curNode->children[str[j] - 'a'];

}

else{

if(curNode->fail == NULL){

curNode = root;

} else{

curNode = curNode->fail;

j--;

}

}

if(curNode->ifEnd){

break;

}

}

if(curNode->ifEnd)

cout << "YES\n";

else

cout << "NO\n";

}

return 0;

}

Game-Theory

*if* p == 'R':

        answer ^= x^y

*if* p == 'B':

        answer ^= min(x,y)

*if* p == 'K':

        temp = 0

*if* min(x,y) % 2 == 0:

*if* max(x,y) % 2 == 0:

                temp = 0

*else*:

                temp = 1

*else*:

*if* max(x,y) % 2 == 0:

                temp = 3

*else*:

                temp = 2

        answer ^= temp

*if* p == 'N':

        a = (x - min(x,y) + min(x,y)  % 3)

        b = (y - min(x,y) + min(x,y)  % 3)

        c = (a+b) // 3

        answer ^= min(a,b,c)

*if* p == 'P':

        answer ^= ((x+y) % 3 + ((x//3)^(y//3)) \* 3)

Slope Trick

*import* heapq

n = int(input())

a = list(map(int, input().split()))

a = [a[i]-i *for* i *in* range(n)]

q = []

heapq.heappush(q, -a[0])

ans = 0

R = [0 *for* \_ *in* range(n)]

R[0] = a[0]

*for* i *in* range(1, n):

    heapq.heappush(q, -a[i])

    R[i] = -q[0] + i

*if* a[i] < -q[0]:

        ans += -q[0] - a[i]

        heapq.heappop(q)

        heapq.heappush(q, -a[i])

*for* i *in* range(n-2, -1, -1):

    R[i] = min(R[i], R[i+1] - 1)

print(\*R, *sep*='\n')

DNC Optimization

import sys

L, G = map(int, sys.stdin.readline().split())

C = list(map(int, sys.stdin.readline().split()))

DP = [[0 for \_ in range(L + 1)] for \_ in range(G + 1)]

CC = [0 for \_ in range(L + 1)]

CC[1] = C[0]

for i in range(2, L + 1):

CC[i] = C[i - 1] + CC[i - 1]

def dnc\_dp(i, s, e, l, r):

if s > e:

return

m = (s + e) // 2

opt = l

mini = DP[i - 1][m]

for k in range(l, m + 1):

if mini >= DP[i - 1][k] + (CC[m] - CC[k]) \* (m - k):

mini = DP[i - 1][k] + (CC[m] - CC[k]) \* (m - k)

opt = k

DP[i][m] = mini

dnc\_dp(i, s, m - 1, l, opt)

dnc\_dp(i, m + 1, e, opt, r)

for i in range(1, L + 1):

DP[1][i] = CC[i] \* i

for i in range(2, G + 1):

dnc\_dp(i, 1, L, 1, L)

print(DP[G][L])