Contents

1 Basic

1.1 Python Test

```
n = 1000000

def updatebit(BIT, n, i, v):
  i += 1
  while i <= n:
    BIT[i] += v
    i += i & (-i)

BITTree = [0]*(n+1)

for i in range(n):
  updatebit(BITTree, n, i, i)
```

1.2 vimrc

```
syn on
se mouse=a ai sta et nu
se ts=2 sts=2 sw=2 st=2 ls=2
ino ( ()<LEFT>
ino [ []<LEFT>
ino ' ''<LEFT>
ino " ""<LEFT>
ino {<CR> {<CR>;<BS><CR>}<UP><TAB>
no <F9> :!g++ -O2 -std=c++17 -lm % -fsanitize=undefined
      -Wall -Wextra -Wshadow -Wno-unused-result<CR>
no <F5> :!./a.out<CR>
```

1.3 Default code

```
#include <bits/stdc++.h>

using namespace std;

#define int long long

void solve() {

}

signed main() {
  ios::sync_with_stdio(0);cin.tie(0);
  int T = 1;
  // cin >> T;
  while (T--) solve();
  return 0;
}
```

1.4 Check

```
for i in $(seq 1 10000);
do
  python3 gen.py
  ./ac < test.in > out_ac
  ./wa < test.in > out_wa
  diff out_ac out_wa || break
done
```

1.5 Black Magic

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
using namespace __gnu_cxx;
typedef tree<int, null_type,less<int>,rb_tree_tag
     ,tree_order_statistics_node_update> TREE;
//tree.find_by_order() / tree.order_of_key();
```

1.6 C++ Random

```
mt19937 gen(chrono
     ::system_clock::now().time_since_epoch().count());
uniform_int_distribution
     <long long int> dist(1, 1000000000000000000);
// usage: dist(gen)
```

2 Bitwise Trick

2.1 Builtin Function

```
// count left 0s
int __builtin_clz
      (unsigned int x) // 31 - __builtin_clz is lg
int __builtin_clzll (unsigned long long x) // 63 - clz
// count number of 1's
int __builtin_popcount (unsigned int x)
int __builtin_popcountll (unsigned long long x)
```

2.2 Next Permutation

```
ll next_perm(ll v) {
  ll t = v | (v - 1);
  return (t + 1) | (((~t & -~t) - 1) >>
             (__builtin_ctz(v) + 1));
}
```

2.3 Subset Enumeration

```
int subset_enumeration(int s) {
  for (int now = s; now > 0; now = (now - 1) & s) {
    cout << now << ' ';
  }
  cout << "0\n";
}
```

3 STL

3.1 Bitset

```
// bit access
b[1] // asscess bit
b.count() // return number of 1
b.size() // return length
b.any() // true if at least one of the bits is set
b.none() // true if none of the bits is set
b.all() // true if all of the bits are set
// bit manipulation
b.to_string(ZERO, ONE) // ZERO: character
     to use to represent false, default '0', ONE
    ; character to use to represent true, default '1'
b.to_ulong()
b.to_ullong()
```

4 Data Structure

4.1 Discrete Trick

```
vector<int> val;
#define ALL(v) (v).begin(), (v).end()
// build
sort(ALL
    (val)), val.resize(unique(ALL(val)) - val.begin());
// index of x
upper_bound(ALL(val), x) - val.begin();
// max idx <= x
upper_bound(ALL(val), x) - val.begin();
// max idx < x
lower_bound(ALL(val), x) - val.begin();
```

4.2 Sparse Table

```
// logn = ceil(log2(mxn))
int st[mxn][logn];
     // sparse table, store answer for [i, i + 2^j - 1]
int a[mxn]; // array
int lg[mxn]; // log value
int n;

void init()
{
  for (int i = 0; i < n; i++) cin >> st[i][0];
  for (int j = 1; j < logn; j++) {
    for (int i = 0; i + (1 << j) <= n; i++){
      st[i][j] = min(
          st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
  }
}
/*
  cin >> n;
  init();
  int l, r; // 1-based
  cin >> l >> r;
  l--, r--;
  int j = __lg(r -
      l + 1); -> min(st[l][j], st[r - (1 << j) + 1][j])
*/
```

4.3 Treap

```
int rnd() { return ((rand
    () % (1 << 15)) << 16) + (rand() % (1 << 15)); }

struct Treap{
  Treap *lc, *rc;
  int sz, pri, val;
  long long int sum;
  bool rev;
  Treap(int _val): lc(nullptr), rc(nullptr), sz(1)
      , pri(rnd()), val(_val), sum(_val), rev(false){};
};
long long int SUM(Treap* a){
  return a? a->sum : 0;
}
int SZ(Treap *a){
  return a? a->sz:0;
}
void pull(Treap* a){
  if(!a){
    return;
  }
  a->sz = 1 + SZ(a->lc) + SZ(a->rc);
  a->sum = a->val + SUM(a->lc) + SUM(a->rc);
}
void push(Treap* a){
  if(a->rev){
    swap(a->lc, a->rc);
  }
  if(a && a->lc){
    a->lc->rev ^= a->rev;
  }
  if(a && a->rc){
    a->rc->rev ^= a->rev;
  }
  a->rev = 0;
}
Treap* merge(Treap* a, Treap* b){
  if(!a || !b)return a? a:b;
  push(a);
  push(b);
  if(a->pri > b->pri){
    a->rc = merge(a->rc, b);
    pull(a);
    return a;
  }
  else{
    b->lc = merge(a, b->lc);
    pull(b);
    return b;
  }
}
void split(Treap* t, int k, Treap* &a, Treap* &b){
  if(!t){
    a = b = nullptr;
    return;
  }
  push(t);
  if(SZ(t->lc)+1 <= k){
    a = t;
    split(t->rc, k-(SZ(t->lc)+1), a->rc, b);
    pull(a);
    return;
  }
  b = t;
  split(t->lc, k, a, b->lc);
  pull(b);
  return;
}
```

4.4 Disjoint Set Undo ver.

```
//parent O(lg(N)), setUp O(lg(N)), undo O(1)
#define MX 10000
int rp[MX],sz[MX];
int compo;
int pts[MX*2],in=0;

int parent(int n){
  if(rp[n]==n)return n;
  return rp[n]=parent(rp[n]);
}

// additionally storing
     parent which is connected to another parents
void setUp(int a,int b){
  a = parent(a);
  b = parent(b);
  if(a==b){
    pts[++in]=-1;
    return;
  }
  if(sz[a]<sz[b]){
    rp[a] = rp[b];
    sz[b] += sz[a];
    pts[++in]=a;
  }
  else{
    rp[b] = rp[a];
    sz[a] += sz[b];
    pts[++in] = b;
  }
  compo--;
}

void undo(){
  if(!in) return;
  int n = pts[in--];
  if(n!=-1) {
    sz[parent(rp[n])] -= sz[n];
    rp[n]=n;
    compo++;
  }
}

void init(int n){
  in=0;
  for(int i=0;i<=MX;i++){
    rp[i]=i;
    sz[i]=1;
  }
  compo=n;
}
```

4.5 Trie

```
class Trie {
public:
  struct Node {
    bool end;
    Node *child[26];
    Node() {
        end = false;
        for (int i = 0; i < 26; i++) child[i] = NULL;
    }
  };
  Node *root;
  Trie() {
    root = new Node();
  }

  void insert(string word) {
    Node *node = root;
    for (char c : word) {
      int ind = c - 'a';
      if (node->child
          [ind] == NULL) node->child[ind] = new Node();
      node = node->child[ind];
    }
    node->end = true;
  }

  bool search(string word) {
    Node *node = root;
    for (char c : word) {
      int ind = c - 'a';
      if (node->child[ind] == NULL) return false;
      node = node->child[ind];
    }
    return node->end;
  }

  bool startsWith(string prefix) {
    Node *node = root;
    for (char c : prefix) {
      int ind = c - 'a';
      if (node->child[ind] == NULL) return false;
      node = node->child[ind];
    }
    return true;
  }
};
```

4.6 Segment Tree(Range chmin, chmax, add)

```
const ll INF = 1e12;
ll a[maxN];
struct Node {
  ll sum;
  ll mx1, mx2, mxc;
  ll mn1, mn2, mnc;
  ll lazy;
  Node () {};
  Node (ll x) : sum(x), mx1(x), mx2(-INF
      ), mn1(x), mn2(INF), mnc(1), mxc(1), lazy(0) {};
} st[maxN << 2];
void Merge(int id) {
  st[id].sum = st[id << 1].sum + st[id << 1 | 1].sum;
  if (st[id << 1].mx1 == st[id << 1 | 1].mx1) {
    st[id].mx1 = st[id << 1].mx1;
    st[id].mx2
         = max(st[id << 1].mx2, st[id << 1 | 1].mx2);
    st[id].mxc = st[id << 1].mxc + st[id << 1 | 1].mxc;
  } else if (st[id << 1].mx1 > st[id << 1 | 1].mx1) {
    st[id].mx1 = st[id << 1].mx1;
    st[id].mx2
         = max(st[id << 1].mx2, st[id << 1 | 1].mx1);
    st[id].mxc = st[id << 1].mxc;
  } else {
    st[id].mx1 = st[id << 1 | 1].mx1;
    st[id].mx2
         = max(st[id << 1].mx1, st[id << 1 | 1].mx2);
    st[id].mxc = st[id << 1 | 1].mxc;
  }
  if (st[id << 1].mn1 == st[id << 1 | 1].mn1) {
    st[id].mn1 = st[id << 1].mn1;
    st[id].mn2
         = min(st[id << 1].mn2, st[id << 1 | 1].mn2);
    st[id].mnc = st[id << 1].mnc + st[id << 1 | 1].mnc;
  } else if (st[id << 1].mn1 < st[id << 1 | 1].mn1) {
    st[id].mn1 = st[id << 1].mn1;
    st[id].mn2
         = min(st[id << 1].mn2, st[id << 1 | 1].mn1);
    st[id].mnc = st[id << 1].mnc;
  } else {
    st[id].mn1 = st[id << 1 | 1].mn1;
    st[id].mn2
         = min(st[id << 1].mn1, st[id << 1 | 1].mn2);
    st[id].mnc = st[id << 1 | 1].mnc;
  }
}
void build(int id, int l, int r) {
  if (l == r) {
    st[id] = Node(a[l]);
    st[id].lazy = 0;
    return;
  }
  int mid = l + r >> 1;
  build(id << 1, l, mid);
  build(id << 1 | 1, mid + 1, r);
  Merge(id);
  st[id].lazy = 0;
}
void push_max(int id, ll val, bool ok) {
  if (val >= st[id].mx1) return;
  st[id].sum -= st[id].mx1 * st[id].mxc;
  st[id].mx1 = val;
  st[id].sum += st[id].mx1 * st[id].mxc;
  if (ok) {
    st[id].mn1 = st[id].mx1;
  } else {
    if (val <= st[id].mn1) {
        st[id].mn1 = val;
    } else if (val < st[id].mn2) {
        st[id].mn2 = val;
    }
  }
}
void push_min(int id, ll val, bool ok) {
  if (val <= st[id].mn1) return;
  st[id].sum -= st[id].mn1 * st[id].mnc;
  st[id].mn1 = val;
  st[id].sum += st[id].mn1 * st[id].mnc;
  if (ok) st[id].mx1 = st[id].mn1;
  else {
    if (val >= st[id].mx1) {
      st[id].mx1 = val;
    } else if (val > st[id].mx2) st[id].mx2 = val;
  }
}
void push_add(int id, ll val, int l, int r) {
  if (val == 0) return;
  st[id].sum += 1ll * (r - l + 1) * val;
  st[id].mx1 += val;
  st[id].mn1 += val;
  if (st[id].mx2 != -INF) st[id].mx2 += val;
  if (st[id].mn2 != INF) st[id].mn2 += val;
  st[id].lazy += val;
}
void down(int id, int l, int r) {
  if (l == r) return;
  int mid = l + r >> 1;
  push_add(id << 1, st[id].lazy, l, mid);
  push_add(id << 1 | 1, st[id].lazy, mid + 1, r);
  st[id].lazy = 0;
  push_max(id << 1, st[id].mx1, l == mid);
  push_max(id << 1 | 1, st[id].mx1, mid + 1 == r);
  push_min(id << 1, st[id].mn1, l == mid);
  push_min(id << 1 | 1, st[id].mn1, mid + 1 == r);
}
```

```
void update_chmin
    (int id, int l, int r, int u, int v, ll val) {
  if (u > r || v < l || st[id].mx1 <= val) return;
  if (u <= l && r <= v && val > st[id].mx2) {
    push_max(id, val, l == r);
    return;
  }
  int mid = l + r >> 1;
  down(id, l, r);
  update_chmin(id << 1, l, mid, u, v, val);
  update_chmin(id << 1 | 1, mid + 1, r, u, v, val);
  Merge(id);
}
void update_chmax
    (int id, int l, int r, int u, int v, ll val) {
  if (u > r || v < l || st[id].mn1 >= val) return;
  if (u <= l && r <= v && st[id].mn2 > val) {
    push_min(id, val, l == r);
    return;
  }
  int mid = l + r >> 1;
  down(id, l, r);
  update_chmax(id << 1, l, mid, u, v, val);
  update_chmax(id << 1 | 1, mid + 1, r, u, v, val);
  Merge(id);
}
void update_add
    (int id, int l, int r, int u, int v, ll val) {
  if (u > r || v < l) return;
  if (u <= l && r <= v) {
    push_add(id, val, l, r);
    return;
  }
  int mid = l + r >> 1;
  down(id, l, r);
  update_add(id << 1, l, mid, u, v, val);
  update_add(id << 1 | 1, mid + 1, r, u, v, val);
  Merge(id);
}
ll get(int id, int l, int r, int u, int v) {
  if (u > r || v < l) return 0;
  if (u <= l && r <= v) return st[id].sum;
  int mid = l + r >> 1;
  down(id, l, r);
  return get(id << 1, l,
       mid, u, v) + get(id << 1 | 1, mid + 1, r, u, v);
}
```

4.7 2D Segment Tree

```
struct SegTree {
    size_t n;
    vector<int64_t> data;
    vector<int> xs;
    void reserve(int i) { xs.push_back(i); }
    void build0() {
        sort(xs.begin(), xs.end());
        for (n = 1; n < xs.size(); n <<= 1);
        data.resize(n << 1);
    }
    void build1() {
        for (int i = n; --i; )
             data[i] = data[i << 1] + data[i << 1 | 1];
    }
    void add_pre(int i, int val) {
        i = lower_bound
            (xs.begin(), xs.end(), i) - xs.begin();
        data[i + n] += val;
    }
    void add(int i, int val) {
        i = lower_bound
            (xs.begin(), xs.end(), i) - xs.begin();
        for (i += n; i; i >>= 1) data[i] += val;
    }
    int64_t sum(int l, int r) {
        l = lower_bound
            (xs.begin(), xs.end(), l) - xs.begin();
        r = lower_bound
            (xs.begin(), xs.end(), r) - xs.begin();
        int64_t res = 0;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (r & 1) res += data[--r];
            if (l & 1) res += data[l++];
        }
        return res;
    }
};
struct SegTree2D {
    size_t n;
    vector<SegTree> trees;
    SegTree2D (size_t n_) {
        for (n = 1; n < n_; n <<= 1);
        trees.resize(n << 1);
    }
    void reserve(int i, int j)
         { // 1. for all node(x, y), call reserve(x, y)
        for (i += n; i; i >>= 1) trees[i].reserve(j);
    }
    void build0() { // 2.
        for (auto &i : trees) i.build0();
    }
    void build1() { // 4.
        for (auto &i : trees) i.build1();
    }
    void add_pre(int i, int j, int val) { // 3.
        for (i
            += n; i; i >>= 1) trees[i].add_pre(j, val);
    }
    // operations: add / sum
    void add(int i, int j, int val) {
        for (i += n; i; i >>= 1) trees[i].add(j, val);
    }
    int64_t sum(int l0, int r0, int
         l1, int r1) {// x in [l0, r0), y in [l1, r1)
        int64_t res = 0;
        for (l0 +=
             n, r0 += n; l0 < r0; l0 >>= 1, r0 >>= 1) {
            if (r0 & 1) res += trees[--r0].sum(l1, r1);
            if (l0 & 1) res += trees[l0++].sum(l1, r1);
        }
        return res;
    }
};
```

4.8 Li Chao Segment Tree

```
using ll = long long;
constexpr ll TEN
    (int n) { return (n == 0) ? 1 : 10 * TEN(n - 1); }

constexpr ll INF = 3 * TEN(18);
struct LiChaoTree {
    using L = pair<ll, ll>;  // l.first * x + l.second
    int sz;
    vector<L> data;
    vector<ll> xs;
    static ll eval
        (L l, ll x) { return l.first * x + l.second; }
    LiChaoTree(vector<ll> _xs) : xs(_xs
        ) { // xs stores all x appears in this problem
        int n = int(xs.size());
        int lg = 1;
        while ((1 << lg) < n) lg++;
        sz = 1 << lg;
        while
            (int(xs.size()) < sz) xs.push_back(TEN(9));
        data = vector<L>(2 * sz, L(0, 3 * TEN(18)));
    }
    void add(L line, int l, int r) { // add({a, b},
         l, r): add a segment f(x) = a*x+b, x in [l, r]
        l = lower_bound
            (xs.begin(), xs.end(), l) - xs.begin();
        r = lower_bound
            (xs.begin(), xs.end(), r) - xs.begin();
        add(line, l, r, 0, sz, 1);
    }
    ll query(ll
         x) { // query(a) : find the minimal y when x=a
        int k = int(lower_bound
            (xs.begin(), xs.end(), x) - xs.begin());
        k += sz;
        ll ans = INF;
        while (k >= 1) {
            ans = min(ans, eval(data[k], x));
            k >>= 1;
        }
        return ans;
    }

  private:
    void add
        (L line, int ql, int qr, int l, int r, int k) {
```

```
        if (qr <= l || r <= ql) {
            return;
        } else if (ql <= l && r <= qr) {
            int mid = (l + r) >> 1;
            ll mx = xs[mid];
            if (eval(line, mx) < eval(data[k], mx)) {
                swap(line, data[k]);
            }
            if (l + 1 == r) return;
            if (line.first > data[k].first) {
                add(line, ql, qr, l, mid, 2 * k);
            } else if (line.first < data[k].first) {
                add(line, ql, qr, mid, r, 2 * k + 1);
            }
        } else {
            int mid = (l + r) >> 1;
            add(line, ql, qr, l, mid, 2 * k);
            add(line, ql, qr, mid, r, 2 * k + 1);
        }
    }
};
```

5 Graph

5.1 Bellman Ford

```
// O(nm)
const ll inf = 1e18;
const int N = 2500 + 5;
int par[N];
ll dist[N];

struct Edge {
  int u, v, w;
};

vector<Edge> edges;

vector<int> Bellman_Ford(int n) {
  for (int i = 1; i <= n; i++) {
    par[i] = -1;
    dist[i] = inf;
  }
  dist[1] = 0ll;
  int x = -1;
  for (int i = 1; i <= n; i++) {
    x = -1;
    for (Edge e : edges) {
      if (dist[e.v] > dist[e.u] + (ll)e.w) {
        dist[e.v] = max(-inf, dist[e.u] + (ll)e.w);
        par[e.v] = e.u;
        x = e.v;
      }
    }
  }
  if (x == -1) return vector<int>();

  for (int i = 1; i <= n; i++)  x = par[x];
  vector<int> cycle;
  for (int u = x;; u = par[u]) {
    cycle.push_back(u);
    if (u == x && SZ(cycle) > 1) break;
  }
  reverse(cycle.begin(), cycle.end());
  return cycle;
}
```

5.2 SPFA

```
// O(m)
const int mxn = 10000 + 5;

bitset<mxn> inque;
vector<pii> g[mxn];
queue<int> q;
vector<ll> dis(mxn, (1ll << 31) - 1);
vector<ll> cnt(mxn, 0);

bool SPFA(int st) {
  q.emplace(st);
  dis[st] = 0;

  while (!q.empty())
  {
    int u = q.front(); q.pop();
    inque[u] = 0;
    for (auto [v, w] : g[u])
    {
      if (dis[v] > dis[u] + w) {
        if (++cnt[v] >=
            n) return false; // contains negative cycle
        dis[v] = dis[u] + w;

        if (!inque[v]) {
          inque[v] = 1;
          q.emplace(v);
        }
      }
    }
  }

  return true;
}
```

5.3 Floyd Warshall

```
const int mxn = 100 + 6;
ll dis[mxn][mxn];

void Floyd()
{
  for (int k = 1; k <= n; k++)
  {
    for (int i = 1; i <= n; i++)
    {
      for (int j = 1; j <= n; j++)
      {
        dis[i][
            j] = min(dis[i][j], dis[i][k] + dis[k][j]);
      }
    }
  }
}
```

5.4 Bi-CC (store vertex)

```
//  bridge is
     defined as an edge which, when removed, makes the
     graph disconnected (or more precisely, increases
     the number of connected components in the graph)
vector<int> G[N]; // 1-base
vector<int> nG[N], bcc[N];
int low[N], dfn[N], Time;
int bcc_id[N], bcc_cnt; // 1-base
bool is_cut[N]; // whether is av
bool cir[N];
int st[N], top;

void dfs(int u, int pa = -1) {
  int child = 0;
  low[u] = dfn[u] = ++Time;
  st[top++] = u;
  for (int v : G[u])
    if (!dfn[v]) {
      dfs(v, u), ++child;
      low[u] = min(low[u], low[v]);
      if (dfn[u] <= low[v]) {
        is_cut[u] = 1;
        bcc[++bcc_cnt].clear();
        int t;
        do {
          bcc_id[t = st[--top]] = bcc_cnt;
          bcc[bcc_cnt].push_back(t);
        } while (t != v);
        bcc_id[u] = bcc_cnt;
        bcc[bcc_cnt].pb(u);
      }
    } else if (dfn[v] < dfn[u] && v != pa)
      low[u] = min(low[u], dfn[v]);
  if (pa == -1 && child < 2) is_cut[u] = 0;
}

void bcc_init(int n) {
  Time = bcc_cnt = top = 0;
  for (int i = 1; i <= n; ++i)
    G[i].clear(), dfn[i] = bcc_id[i] = is_cut[i] = 0;
}

void bcc_solve(int n) {
  for (int i = 1; i <= n; ++i)
    if (!dfn[i]) dfs(i);
  // block-cut tree
  for (int i = 1; i <= n; ++i)
    if (is_cut[i])
      bcc_id[i] = ++bcc_cnt, cir[bcc_cnt] = 1;
  for (int i = 1; i <= bcc_cnt && !cir[i]; ++i)
    for (int j : bcc[i])
      if (is_cut[j])
        nG[i].pb(bcc_id[j]), nG[bcc_id[j]].pb(i);
}
```

5.5 Bi-CC (store edge)

```
// bridge is
     defined as an edge which, when removed, makes the
     graph disconnected (or more precisely, increases
     the number of connected components in the graph)
int low[N], dfn[N];
bool vis[N];
int e[M], x[M], y[M]; // e[i] = x[i] ^ y[i]
int stamp, bcc_no = 0;

vector<int> G[N], bcc[N]; // 1-base
stack<int> sta;
int bcc_id[M]; // edge i belongs to bcc_id[i], 1-base

void add_edge(int a, int b, int id){
  G[a].push_back(id);
  G[b].push_back(id);
  x[id] = a;
  y[id] = b;
  e[id] = a ^ b;
}

void dfs(int now, int par_eid) {
  vis[now] = true;
  dfn[now] = low[now] = (++stamp);
  for (int i : G[now]) {
    if (i == par_eid) continue;
    int to = (e[i] ^ now);
    if (!vis[to]) {
      sta.push(i); dfs(to, i);
      low[now] = min(low[now], low[to]);
      if (low[to] >= dfn[now]) {
        ++bcc_no; int p; // p is edge index
        do {
          p = sta.top(); sta.pop();
          bcc_id[p] = bcc_no;
          bcc[bcc_no].push_back(p);
        } while (p != i);
      }
    }
    else if (dfn[to] < dfn[now]) {
      sta.push(i);
      low[now] = min(low[now], dfn[to]);
    }
  }
}

void bcc_solve(int n) {
  for (int i = 1; i <= n; ++i)
    if (!dfn[i]) dfs(i, -1);
}

// add_edge -> bcc_solve
// record nodes in bcc:
// for(int i = 1; i <= bcc_no; i++){
//  for(auto eid:bcc[i]){
//    bcc_node[i].insert(eS[eid].X);
//    bcc_node[i].insert(eS[eid].Y);
//  }
// }
//  pii eS[M]: edge i connect eS[i].X and eS[i].Y
//  set<int> bcc_node[N]: bcc i has bcc_node[i] nodes
```

5.6 Bridge-CC

```
int bcc[MAXN
    ], dic[MAXN], low[MAXN], component = 0, times = 1;
vector<int> Stack;

void init(){
  Stack.clear();
  memset(bcc,-1, sizeof(int) * MAXN);
  memset(dic,0, sizeof(int) * MAXN);
  memset(low,0, sizeof(int) * MAXN);
  component = 0;
  times = 1;
}

void tarjan(vector<int> *adj, int start, int parent){
  dic[start] = low[start] = times; times ++;
  Stack.push_back(start);
  for(auto v : adj[start]){
    if (dic[v] == 0)
      tarjan(adj,v, start);
    if (v != parent)
      low[start] = min(low[start], low[v]);
  }

  if(low[start] == dic[start]){
    int t = 0;
    do {
      t = Stack.back();
      bcc[t] = component;
      Stack.pop_back();
    } while(t != start);
    component++;
  }
}

void findbcc(vector<int> *adj, int N){
  init();
  for(int i = 0; i < N; i++){
    if(dic[i] == 0)
      tarjan(adj, i, i);
  }
}

vector<int> build_bcc_graph(vector<int> *adj, int N){
  vector<int> adj_bcc[component];
  for(int l = 0; l < N; l++){
    for(auto j : adj[l]){
        int root1 = bcc[l], root2 = bcc[j];
        if(root1 != root2){
            adj_bcc[root1].push_back(root2);
            adj_bcc[root2].push_back(root1);
        }
    }
  }
}
```

5.7 SCC

```
struct SCC {
  vector<int> g[N];
  vector<int> comp, ind;
  stack<int> sta;
  int di = 0; // DFS counter
  int cc = 0; // Comp count

  void add_edge(int s, int e) {
    g[s].push_back(e);
  }
  int dfs(int i) {
    if (ind[
        i] != -1) return (comp[i] == -1) ? ind[i] : di;
    ind[i] = di;
    int md = di;
    ++di;

    sta.push(i);
    for (auto t : g[i]) md = min(md, dfs(t));

    if (md == ind[i]) {
      while(comp[i] == -1) {
        comp[sta.top()] = cc;
        sta.pop();
      }
      ++cc;
    }
    return md;
  }
  void get() {
    for (int i = 0; i < N; i++) dfs(i);
  }
  SCC() {
    comp.assign(N, -1);
    ind.assign(N, -1);
  }
};
```

5.8 2-SAT

```
using Vi = vector<int>;
#define pb push_back
#define each(a, x) for (auto &a: (x))
struct SAT2 : Vi {
  vector<Vi> G;
  Vi order, flags;

  // Init n variables, you can add more later
  SAT2(int n = 0) : G(n * 2) {}

  // Add new var and return its index
  int addVar() {
    G.resize(SZ(G) + 2); return SZ(G)/2;
  }

  // Add (i => j) constraint
  void imply(int i, int j) {
    i = max(i * 2 - 1, -i * 2 -2);
    j = max(j * 2 - 1, -j * 2 - 2);
    G[i].pb(j); G[j ^ 1].pb(i ^ 1);
  }

  // Add (i v j) constraint
  void either(int i, int j) { imply(-i, j); }

  // Add !(i ▯ j) constraint
  void notBoth(int i, int j) { imply(i, -j); }

  // Constraint at most one true variable
  void atMostOne(Vi& vars) {
    int x = addVar();
    each(i, vars) {
      int y = addVar();
      imply(x, y); imply(i, -x); imply(i, y);
      x = y;
    }
  }

  // Solve and save assignments in `values`
  bool solve() { // O(n+m), Kosaraju is used
    assign(SZ(G)/2+1, -1);
    flags.assign(SZ(G), 0);
    for (int i = 0; i < SZ(G); i++) dfs(i);
    while (!order.empty()) {
      if (!propag(order.back()^1, 1)) return 0;
      order.pop_back();
    }
    return 1;
  }

  void dfs(int i) {
    if (flags[i]) return;
    flags[i] = 1;
    each(e, G[i]) dfs(e);
    order.pb(i);
  }

  bool propag(int i, bool first) {
    if (!flags[i]) return 1;
    flags[i] = 0;
    if (at(i/2+1) >= 0) return first;
    at(i/2+1) = i&1;
    each(e, G[i]) if (!propag(e, 0)) return 0;
    return 1;
  }
};
```

5.9 Kruskal

```
// edge[i] = {z, x, y}
// edge(x, y) with weight z
vector<vector<int>> edgelist;

int Kruskal_MST(int N){
  int set_count = N, weight = 0;
  // make set for all vertex
  disjoint_set pointset[N];
  for(int i = 0; i < N; i++)
    pointset[i].make_set(i);
  // sort edges
  sort(edgelist.begin(), edgelist.end());
  // start union
  for(auto it : edgelist){
    // if in different set, then union
    if(pointset[it
        [1]].find_set() != pointset[it[2]].find_set()){
      Union(&pointset[it[1]], &pointset[it[2]]);
      set_count --;
      weight += it[0];
    }
  }
  if(set_count != 1) return -1;
  return weight;
}
```

5.10 Prim

```
vector<pii> edges[MAXN];
// edges[i]:
    edge(i, edges[i].second) with weight edges[i].first
// O(ElogE)
int Prim_MST(int N){
  bool choose[N] = {0};
  int count = 0, weight = 0;
  priority_queue<pii, vector<pii>, greater<pii>> pq;
  // choose one vertex
  choose[0] = 1;
  count ++;
  // push its edges to pq
  for(auto it : edges[0]){
    pq.push(it);
  }
  // do it until pq empty
  while(!pq.empty()){
    auto add = pq.top();
    pq.pop();
    if(choose[add.second])
      continue;
    choose[add.second] = 1;
    //printf("add:
         vertex %d, with %d\n", add.second, add.first);
    count ++;
    weight += add.first;
    for(auto it : edges[add.second]){
      pq.push(it);
    }
  }
  if(count != N) return -1;
  return weight;
}
```

5.11 Euler Trail

```
// del[now] : G[now][1,2⋯⋯,del[now]-1] are visited
// start from G[now][del[now]]
int del[MAXN] = {0};
// in-degree, out-degree
// in, out degree are calculated when add edges
int in[MAXN] = {0}, out[MAXN] = {0};
stack <int> st;
// G[i] is sorted, represent directed graph
vector <int> G[MAXN];

void dfs(int now)
{
  for(int i=del[now];i<G[now].size();i=del[now])
  {
    del[now]=i+1;
    dfs(G[now][i]);
  }
  st.push(now);
}

void Euler_Path(int N){
  // if euler circuit exists
  // start from first vertex
  int S = 1, T, cnt[2] ={0};
  // in == out
  bool flag = true;
  for(int i = 1; i <= N; i++){
    if(in[i] != out[i]) flag=false;
    if(out[i] - in[i] == 1) cnt[0]++, S=i; // start
    if(in[i] - out[i] == 1) cnt[1]++, T=i; // terminate
  }
  // if needed, we need to check whether there's
  // a path from S to T
  if(!flag && !(cnt[0] == 1 && cnt[1] == 1)){
    printf("No");
    return;
  }
  dfs(S);
  while(!st.empty()) printf("%d ",st.top()),st.pop();
  return;
}
```

5.12 Euler Tour

```
vector<pair<int, int>> g[kN];
vector<int> euler_tour;
vector<bool> visited(kN, false);
void EulerTour(int x){
  while(!g[x].empty()){
    auto [u, e] = g[x].back();//e is the ID of the edge
    g[x].pop_back();
    if(!visited[e]){
      visited[e] = true;
      EulerTour(u);
      euler_tour.push_back(e);
    }
  }
}
```

5.13 AP & Bridge

```
const int mxn = 5e4 + 5;
vector<int> g[mxn];
int cnt = 0;
int dfn[mxn], low[mxn], vis[mxn], ap[mxn];
set<int> nodeAP;
set<pii> bridge;

void tarjan(int u, int fa)
{
  vis[u] = 1;
  low[u] = dfn[u] = ++cnt;
  int child = 0;
  for (int v : g[u])
  {
    if (v == fa) continue;
    if (!dfn[v]) {
      child++;
      tarjan(v, u);
      low[u] = min(low[u], low[v]);
      if (fa != -1 && low[v] >= dfn[u] and !ap[u]) {
        ap[u] = 1;
        nodeAP.insert(u);
      }
      if (low[v] > dfn[u]) {
        if (v < u) bridge.insert({v, u});
        else bridge.insert({u, v});
      }
    }
    else low[u] = min(low[u], dfn[v]);
  }

  if (fa == -1 && child > 1 && !ap[u]) {
    ap[u] = 1;
    nodeAP.insert(u);
  }
}

void solve()
{
  for (int i = 1; i <= n; i++)
  {
    if (!dfn[i]) {
      tarjan(i, -1);
    }
  }
}
```

5.14 Max Clique

```
const int N = 45;
struct MaxClique {
  bitset<N> a[N], cs[N];
  int ans, sol[N], q, cur[N], d[N], n;
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; i++) a[i].reset();
  }
  void addEdge(int u, int v) { a[u][v] = a[v][u] = 1; }
  void csort(vector<int> &r, vector<int> &c) {
    int mx = 1, km = max(ans - q + 1, 1), t = 0,
        m = r.size();
    cs[1].reset(), cs[2].reset();
    for (int i = 0; i < m; i++) {
      int p = r[i], k = 1;
      while ((cs[k] & a[p]).count()) k++;
      if (k > mx) mx++, cs[mx + 1].reset();
      cs[k][p] = 1;
      if (k < km) r[t++] = p;
    }
    c.resize(m);
    if (t) c[t - 1] = 0;
    for (int k = km; k <= mx; k++)
      for (int p = cs[k]._Find_first(); p < N;
           p = cs[k]._Find_next(p))
        r[t] = p, c[t] = k, t++;
  }
  void dfs(vector<int> &r, vector<int> &c, int l,
    bitset<N> mask) {
    while (!r.empty()) {
      int p = r.back();
      r.pop_back(), mask[p] = 0;
      if (q + c.back() <= ans) return;
      cur[q++] = p;
      vector<int> nr, nc;
      bitset<N> nmask = mask & a[p];
      for (int i : r)
        if (a[p][i]) nr.push_back(i);
      if (!nr.empty()) {
        if (l < 4) {
          for (int i : nr)
            d[i] = (a[i] & nmask).count();
          sort(nr.begin(), nr.end(),
            [&](int x, int y) { return d[x] > d[y]; });
        }
        csort(nr, nc), dfs(nr, nc, l + 1, nmask);
      } else if (q > ans) ans = q, copy_n(cur, q, sol);
      c.pop_back(), q--;
    }
  }
  int solve(bitset<N> mask = bitset<N>(
              string(N, '1'))) {
    vector<int> r, c;
    ans = q = 0;
    for (int i = 0; i < n; i++)
      if (mask[i]) r.push_back(i);
    for (int i = 0; i < n; i++)
      d[i] = (a[i] & mask).count();
    sort(r.begin(), r.end(),
      [&](int i, int j) { return d[i] > d[j]; });
    csort(r, c), dfs(r, c, 1, mask);
    return ans; // sol[0 ~ ans-1]
  }
} graph;
```

5.15 Vizing

```
namespace vizing { // returns
     edge coloring in adjacent matrix G. 1 - based
const int N = 105;
int C[N][N], G[N][N], X[N], vst[N], n;
void init(int _n) { n = _n;
  for (int i = 0; i <= n; ++i)
    for (int j = 0; j <= n; ++j)
      C[i][j] = G[i][j] = 0;
}
void solve(vector<pii> &E) {
  auto update = [&](int u)
  { for (X[u] = 1; C[u][X[u]]; ++X[u]); };
  auto color = [&](int u, int v, int c) {
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v, C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if (p) X[u] = X[v] = p;
    else update(u), update(v);
    return p;
  };
  auto flip = [&](int u, int c1, int c2) {
    int p = C[u][c1];
    swap(C[u][c1], C[u][c2]);
    if (p) G[u][p] = G[p][u] = c2;
    if (!C[u][c1]) X[u] = c1;
    if (!C[u][c2]) X[u] = c2;
    return p;
  };
  fill_n(X + 1, n, 1);
  for (int t = 0; t < SZ(E); ++t) {
    int u = E[t
        ].X, v0 = E[t].Y, v = v0, c0 = X[u], c = c0, d;
    vector<pii> L;
    fill_n(vst + 1, n, 0);
    while (!G[u][v0]) {
      L.emplace_back(v, d = X[v]);
      if (!C[v][c]) for (int a = SZ(
          L) - 1; a >= 0; --a) c = color(u, L[a].X, c);
      else if (!C[u][d]) for (int a = SZ(L
          ) - 1; a >= 0; --a) color(u, L[a].X, L[a].Y);
      else if (vst[d]) break;
      else vst[d] = 1, v = C[u][d];
    }
    if (!G[u][v0]) {
      for (; v; v = flip(v, c, d), swap(c, d));
      if (int a; C[u][c0]) {
        for (
            a = SZ(L) - 2; a >= 0 && L[a].Y != c; --a);
        for (; a >= 0; --a) color(u, L[a].X, L[a].Y);
      }
      else --t;
    }
  }
}
} // namespace vizing
```

5.16 Dominator Tree

```
struct dominator_tree { // 1-base
  vector<int> G[N], rG[N];
  int n, pa[N], dfn[N], id[N], Time;
  int semi[N], idom[N], best[N];
  vector<int> tree[N]; // dominator_tree
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; ++i)
      G[i].clear(), rG[i].clear();
  }
  void add_edge(int u, int v) {
    G[u].pb(v), rG[v].pb(u);
  }
  void dfs(int u) {
    id[dfn[u] = ++Time] = u;
    for (auto v : G[u])
      if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
  }
  int find(int y, int x) {
    if (y <= x) return y;
    int tmp = find(pa[y], x);
    if (semi[best[y]] > semi[best[pa[y]]])
      best[y] = best[pa[y]];
    return pa[y] = tmp;
  }
  void tarjan(int root) {
    Time = 0;
    for (int i = 1; i <= n; ++i) {
      dfn[i] = idom[i] = 0;
      tree[i].clear();
      best[i] = semi[i] = i;
    }
    dfs(root);
    for (int i = Time; i > 1; --i) {
      int u = id[i];
      for (auto v : rG[u])
        if (v = dfn[v]) {
          find(v, i);
          semi[i] = min(semi[i], semi[best[v]]);
        }
      tree[semi[i]].pb(i);
      for (auto v : tree[pa[i]]) {
        find(v, pa[i]);
        idom[v] =
          semi[best[v]] == pa[i] ? pa[i] : best[v];
      }
      tree[pa[i]].clear();
    }
    for (int i = 2; i <= Time; ++i) {
      if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
      tree[id[idom[i]]].pb(id[i]);
    }
  }
} dt;
// init -> add_edge -> tarjan(root)
// tree: u -> v, from root to v must pass u
```

6 String

6.1 KMP

```
// build pi function
// pi(i) is
     the length of the longest prefix of the substring
     s[0..i] which is also a suffix of this substring.
vector<int> prefix_function(string s) {
    int n = SZ(s);
    vector<int> pi(n, 0);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}

// find all pattern occurence starting index in text
vector<int> KMP(string text, string pat) {
  string s = pat + "#" + text;
  vector<int> pi = prefix_function(s);
  vector<int> res;
  int n = SZ(pat);
  for (int i = n + 1; i < SZ(s); i++) {
    if (pi[i] == n) {
      res.push_back(i - 2 * n);
    }
  }
  return res;
}
```

6.2 String Matching bitset ver.

```
// record alphabets' occurence in text
bitset<10000> mask[26];
string text;

// set up mask
void SetMask(){
  text = "#" + text; // let the text become 1-indexed
  for(int i = 1; i < text.size(); i++){
    mask[text[i]-'a'][i] = 1; // set up bit
  }
}

// modified mask and the string
void ChangeMask(int pos, char c){
  mask[text[pos]-'a'][pos] = 0;
  text[pos] = c;
  mask[text[pos]-'a'][pos] = 1;
}

// find pattern count in text[l..r]
int Substring(string pattern, int l, int r){
  // range validation
  if (r - l + 1 < pattern.size())
    return 0;
  // 1. set a ans_mask with all 1
  // 2. for i in pattern
      , ans_mask &= (mask[pattern[i] - 'a'] >> i);
  // 3. ans_mask
       >> (l-1).count - ans_mask >> (r-len+1).count
  bitset<MAX> ans_mask;
  ans_mask.set();
  pattern = "#" + pattern;
  for(int i = 1; i < pattern.size(); i++){
    ans_mask &= (mask[pattern[i] - 'a'] >> i);
  }
  return (ans_mask >> (l-1)).count()
      - (ans_mask >> (r - pattern.size() + 2)).count();
}
```

6.3 Z-value

```
// z[i] =
     s and s[i, n - 1] max Longest Common Prefix length
int z[MAXN];
void make_z(const string &s) {
  int l = 0, r = 0;
  z[0] = SZ(s);
  for (int i = 1; i < SZ(s); ++i) {
    for (z[i] = max(0, min(r - i + 1, z[i - l]));
         i + z[i] < SZ(s) && s[i + z[i]] == s[z[i]];
         ++z[i])
      ;
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
}
```

6.4 Manacher

```
// return length of longest palindrome in string tmp
// remember mxn should be at least 2 * string length
int d[mxn];
int Manacher(string tmp)
{
#define pb push_back
  string s = "&";
  int l = 0, r = 0, x = 0, ans = 0;
  for (char c : tmp) s.pb(c), s.pb('%');
  int sz = s.size();
  for (int i = 1; i < sz; i++)
  {
    d[i] = r > i ? min(d[2 * l - i], r - i) : 1;
    while (s[i + d[i]] == s[i - d[i]]) d[i]++;
    if (d[i] + i > r) r = i + d[i], l = i;
  }
  for (int i = 1; i < sz; i++)
  {
    if (s[i] == '%') x = max(x, d[i]);
  }
  ans = x / 2 * 2, x = 0;
  for (int i = 1; i < sz; i++)
  {
    if (s[i] != '%') x = max(x, d[i]);
  }
  return max(ans, (x - 1) / 2 * 2 + 1);
}

// return longest palindrome in string tmp
vector<int> manacherOdd(string s) {
  int n = SZ(s);
  s = "$" + s + "^";
  vector<int> p(n+2);
  int l = 1, r = 1;
  for (int i = 1; i <= n; i++) {
    p[i] = max(0, min(r - i, p[l + r - i]));
    while (s[i - p[i]] == s[i + p[i]]) p[i]++;
    if (i + p[i] > r) {
      l = i - p[i];
      r = i + p[i];
    }
  }
  return vector<int>(begin(p)+1,end(p)-1);
}

string manacher(string s) {
  string t;
  for(auto c : s) t += string("#") + c;
  vector<int> ans = manacherOdd(t + "#");
  int maxid = 1, maxlen = ans[1];
  for (int i = 1; i < SZ(ans) - 1; i++) {
    if (ans[i] > maxlen) {
      maxlen = ans[i];
      maxid = i;
    }
  }
  return s.
      substr(maxid / 2 - (maxlen - 1) / 2, maxlen - 1);
}
```

6.5 Suffix Array

```
struct suffix_array {
  // 0-indexed
  // box: bucket
  // m: ASCII limi
  int box[MAXN], tp[MAXN], m;
  // sa[ra[i]] = ra[sa[i]] = i;
  // he[i] = LCP(i-1,i);
  int sa[MAXN], ra[MAXN], he[MAXN];

  void radix(int *rk, int *in, int *out, int n) {
    fill_n(box, m, 0);
    for (int i = 0; i < n; ++i) ++box[rk[i]];
    partial_sum(box, box + m, box);
    for (int i = n - 1; i >= 0; --i)
      out[--box[rk[in[i]]]] = in[i];
  }
  bool not_equ(int a, int b, int k, int n) {
    return ra[a] != ra[b] || a + k >= n ||
      b + k >= n || ra[a + k] != ra[b + k];
  }
  void make_sa(const string &s, int n) {
    int k = 1;
    for (int i = 0; i < n; ++i) ra[i] = s[i];
    do {
      iota(tp, tp + k, n - k), iota(sa + k, sa + n, 0);
      radix(ra + k, sa + k, tp + k, n - k);
      radix(ra, tp, sa, n);
      tp[sa[0]] = 0, m = 1;
      for (int i = 1; i < n; ++i) {
        m += not_equ(sa[i], sa[i - 1], k, n);
        tp[sa[i]] = m - 1;
      }
      copy_n(tp, n, ra);
      k *= 2;
    } while (k < n && m != n);
  }
  void make_he(const string &s, int n) {
    for (int j = 0, k = 0; j < n; ++j) {
      if (ra[j])
        for (; s[j + k] == s[sa[ra[j] - 1] + k]; ++k)
          ;
      he[ra[j]] = k, k = max(0, k - 1);
    }
  }
  void build(const string &s) {
    int n = SZ(s);
    fill_n
        (sa, n, 0), fill_n(ra, n, 0), fill_n(he, n, 0);
    fill_n(box, n, 0), fill_n(tp, n, 0), m = 256;
    make_sa(s, n), make_he(s, n);
  }
};
```

6.6 SAIS

```
namespace sfx {
bool _t[N * 2];
int SA[N * 2], H[N], RA[N];
int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2];
// zero based, string content MUST > 0
// SA[i]: SA[i]-th
    suffix is the i-th lexigraphically smallest suffix.
// H[i]: longest
    common prefix of suffix SA[i] and suffix SA[i - 1].
void pre(int *sa, int *c, int n, int z)
{ fill_n(sa, n, 0), copy_n(c, z, x); }
void induce
    (int *sa, int *c, int *s, bool *t, int n, int z) {
  copy_n(c, z - 1, x + 1);
  for (int i = 0; i < n; ++i)
    if (sa[i] && !t[sa[i] - 1])
      sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
  copy_n(c, z, x);
  for (int i = n - 1; i >= 0; --i)
    if (sa[i] && t[sa[i] - 1])
      sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}
void sais(int *s, int *sa
    , int *p, int *q, bool *t, int *c, int n, int z) {
  bool uniq = t[n - 1] = true;
  int nn = 0,
      nmxz = -1, *nsa = sa + n, *ns = s + n, last = -1;
  fill_n(c, z, 0);
  for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
  partial_sum(c, c + z, c);
  if (uniq) {
    for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
    return;
  }
  for (int i = n - 2; i >= 0; --i)
    t[i] = (
        s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i + 1]);
  pre(sa, c, n, z);
  for (int i = 1; i <= n - 1; ++i)
    if (t[i] && !t[i - 1])
      sa[--x[s[i]]] = p[q[i] = nn++] = i;
  induce(sa, c, s, t, n, z);
  for (int i = 0; i < n; ++i)
    if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
      bool neq = last < 0 || !equal
          (s + sa[i], s + p[q[sa[i]] + 1], s + last);
      ns[q[last = sa[i]]] = nmxz += neq;
    }
  sais(ns,
       nsa, p + nn, q + n, t + n, c + z, nn, nmxz + 1);
  pre(sa, c, n, z);
  for (int i = nn - 1; i >= 0; --i)
    sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
  induce(sa, c, s, t, n, z);
}
void mkhei(int n) {
  for (int i = 0, j = 0; i < n; ++i) {
    if (RA[i])
      for (; _s[i + j] == _s[SA[RA[i] - 1] + j]; ++j);
    H[RA[i]] = j, j = max(0, j - 1);
  }
}
void build(int *s, int n) {
  copy_n(s, n, _s), _s[n] = 0;
  sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
  copy_n(SA + 1, n, SA);
  for (int i = 0; i < n; ++i) RA[SA[i]] = i;
  mkhei(n);
}}
```

6.7 Lexicographically Smallest Rotation

```
// return the lexicographically smallest
// string among all rotation
string mcp(string s) {
  #define SZ(a) ((int)a.size())
  int n = SZ(s), i = 0, j = 1;
  s += s;
  while (i < n && j < n) {
    int k = 0;
    while (k < n && s[i + k] == s[j + k]) ++k;
    if (s[i + k] <= s[j + k]) j += k + 1;
    else i += k + 1;
    if (i == j) ++j;
  }
  int ans = i < n ? i : j;
  return s.substr(ans, n);
}
```

6.8 AC Automaton Arr.ver

```
int Node[MAXN][CSIZE] = {{0}}, EndOfWord[MAXN
    ]={0}, id_of_str[MAXN] = {0}, failptr[MAXN] = {0};
int id = 0,
    que[MAXN]={0}, cnt[MAXN] = {0}, end_at[MAXN] = {0};

// usage
// ac.build(a); "a" contains patterns
// ac.find(t); t is the target text
// end_at[i]: a[i] ends at node end_at[i]
// cnt[end_at[i]]: a[i] appears cnt[end_at[i]] in t

int getindex(char a){
  return a - 'a';
}

void init(){
  memset(Node, 0, sizeof(Node));
  memset(EndOfWord, 0, sizeof(EndOfWord));
  memset(id_of_str, 0, sizeof(id_of_str));
  memset(failptr, -1, sizeof(failptr));
  memset(que, 0, sizeof(que));
  memset(cnt, 0, sizeof(cnt));
  id = 0;
}

void insert(string s, int id_){
  int rt = 0, len = s.length();
  for(int i = 0; i < len; i++){
    int index = getindex(s[i]);
    if(!Node[rt][index]){
      id ++;
      Node[rt][index] = id;
    }
    rt = Node[rt][index];
  }
  EndOfWord[rt] ++;
  id_of_str[rt] = id_;
  end_at[id_] = rt;
}

struct AC_Automaton{
  int root = 0;
  vector<string> patterns;

  void build(vector<string> patterns_){
    patterns = patterns_;
    int size = SZ(patterns);
    init();
    // build trie
    for(int i = 0; i < size; i++){
      insert(patterns[i], i);
    }

    // build failptr
    queue<int> q;
    int t = 0;
    // first layer's failptr is root
    for(int i = 0; i < CSIZE; i++){
      if(Node[root][i]){
        failptr[Node[root][i]] = root;
        q.push(Node[root][i]);
      }
    }

    // BFS
    while(!q.empty()){
      int x = q.front();
      q.pop();
      que[t++] = x;
      for(int i = 0; i < CSIZE; i++){
        if(Node[x][i]){
          q.push(Node[x][i]);
          int Fail = failptr[x];
          while(1){
            if(Fail == -1){
              // root
              failptr[Node[x][i]] = root;
              break;
            }

            if(Node[Fail][i]){
              failptr[Node[x][i]] = Node[Fail][i];
              break;
            } else {
              Fail = failptr[Fail];
            }
          }
        }
      }
    }
  }

  void find(string text){
    memset(cnt, 0, sizeof(cnt));
    int curr = root;
    int i = 0;
    int size = SZ(text);
    while(i < size){
      int index = getindex(text[i]);

      if(Node[curr][index]){
        curr = Node[curr][index];
        cnt[curr] ++;
        i++;
      } else {
        curr = failptr[curr];
        if(curr == -1){
          curr = root;
          cnt[curr] ++;
          i++;
        }
      }
    }
    for(int i = id - 1; i >= 0; i--){
      cnt[failptr[que[i]]] += cnt[que[i]];
    }
  }
};
```

7 Math

7.1 extgcd

```
// find [x, y] such that ax + by = 1;
ll extgcd(ll a, ll b, ll &x, ll &y) {
  if (!b) { x = 1; y = 0; return a; }
  ll d = extgcd(b, a % b, y, x);
  y -= (a / b) * x;
  return d;
}
```

7.2 Chinese Remainder Theorem

```
// x □ b_i
    (mod a_i), a_i is not promised to be mutually prime
ll mul(ll x, ll y, ll m) {
  ll res = 0;
  while (y) {
    if (y & 1) res = (res + x) % m;
    x = (x + x) % m;
    y >>= 1;
  }
  return res;
}
ll extgcd(ll a, ll b, ll &x, ll &y) {
  if (!b) { x = 1; y = 0; return a; }
  ll d = extgcd(b, a % b, y, x);
  y -= (a / b) * x;
  return d;
}
ll excrt() {
  ll clcm = a[0], res = b[0];
  for (int i = 1; i < n; i++) {
    ll x, y, d = extgcd(clcm, a[i], x, y);
    ll r = ((b[i] - res) % a[i] + a[i]) % a[i];
    ll tmp = clcm / d * a[i];
    if (r % d) return -1;
    x = (mul(x, r / d, a[i]) + a[i]) % a[i];
    res = (res + mul(x, clcm, tmp)) % tmp;
    clcm = tmp;
  }
  return res;
}
```

7.3 Modular Multiplicative Inverse

```
ll inv[mxn];

void mod_inv() {
  inv[1] = 1;
  for (ll i = 2; i <= n; i++) {
    inv[i] = (p - p / i) * inv[p % i] % p;
  }
}
```

7.4 Build Prime Table

```
const int N = 1e8 + 5;

bitset<N> num;
vector<int> prime;
void LS(int n) {
  for (int i = 2; i <= n; ++i) {
    if (!num[i])
        prime.push_back(i);
    for (int j = 0; j < prime.size(); ++j) {
      if (i * prime[j] >= n) break;
      num[i * prime[j]] = 1;
      if (i % prime[j] == 0)break;
    }
  }
}
```

7.5 Floor & Ceil

```
int ifloor(int a,int b)
// if a/b < 0 && a%b!=0
// minus 1
{ return a / b - (a % b && a < 0 ^ b < 0); }
int iceil(int a,int b)
// if a/b > 0 && a%b!=0
// plus 1
{ return a / b + (a % b && a < 0 ^ b > 0); }
```

7.6 Gauss Elimation Normal

```
double a[N][N], x[N];
const double eps=1e-6;
int solve(int n, int m){
  int c=0;
  int r;
  for(r = 0; r < n && c < m; r++, c++) {
    int maxr = r;
    for(int i = r + 1; i < n; i++) {
      if(abs(a[i][c]) > abs(a[maxr][c]))
        maxr = i;
    }
    if(maxr != r) swap(a[r], a[maxr]);
    if(fabs(a[r][c]) <eps){
      r--;
      continue;
    }
    for(int i = r + 1; i < n; i++){
      if(fabs(a[i][c]) > eps){
        double k = a[i][c] / a[r][c];
        for(int j =
             c; j < m + 1; j++) a[i][j] -= a[r][j] * k;
        a[i][c] = 0;
      }
    }
  }
  for(int i = r; i < m; i++){
    if(fabs(a[i][c]) > eps) return -1; // no solution
  }
  if(r < m) return 1; // infinite solution
  for(int i = m - 1; i >= 0; i--){
      for(int j =
          i + 1; j < m; j++) a[i][m] -= a[i][j] * x[j];
      x[i] = a[i][m] / a[i][i];
  }
  return 0; // only one solution
}

// usage:
// for i in [0, n) :
//  for j in [0, n] :
//    cin >> a[i][j];
// int res = solve(n, n);
// result in x[0..n - 1]
```

7.7 Gauss Elimation xor

```
int a[N][N];
void Gauss(int n) {
  int c, r;
  for (c = 0, r = 0; c < n; c++) {
    int t = r;
    for (int i = r; i < n; i++) {
      if (a[i][c]) {
        t = i;
        break;
      }
    }
    if (!a[t][c]) continue;
    for (
        int i = c; i <= n; i++) swap(a[r][i], a[t][i]);
    for (int i = r + 1; i < n; i++)
      if (a[i][c])
        for (
            int j = n; j >= c; j--) a[i][j] ^= a[r][j];
    r++;
  }
  if (r < n) {
    for (int i = r; i < n; i++) {
      if (a[i][n]) {
        return -1; // no solution
      }
    }
    return 1; // infinite solution
  }
  for (int i = n - 1; i >= 0; i--)
    for (int j = i + 1; j < n; j++)
      a[i][n] ^= a[i][j] * a[j][n];
  return 0; // only on solution
}

//usage:
//for i in [0, n):
//  for j in [0, n]:
//    cin >> a[i][j];
//int res = Gauss(n);
//result in a[0..n - 1][n]
```

7.8 Gauss Elimation with Rank and Basis

```
// find x of Ax = B
typedef unsigned int ui;
typedef unsigned long long ull;
const ui M = 998244353;

ui mypw(ui x, int y) {
  ui res = 1;
  while (y) {
    if (y & 1) res = (ull)res * x % M;
    x = (ull)x * x % M;
    y >>= 1;
  }
  return res;
}

tuple<int,
    vector<ui>, vector<vector<ui>>> gauss(vector<vector
    <ui>> &a) { // {rank, one of the solution, basis}
  int n = SZ(a), m = SZ(a[0])-1, i, j, k, R=m;
  vector<int> fix(m, -1);
  for (i = k = 0; i < m; i++) {
    for (j = k; j < n; j++) if (a[j][i]) break;
    if (j == n) continue;
    fix[i] = k; --R;
    swap(a[k],a[j]);
    ui *u = a[k].data();
    ui x = mypw(u[i], M - 2);
    for (j = i; j <= m; j++) u[j] = (ull)u[j] * x % M;
    for (auto &v:a) if (v.data() != a[k].data())
    {
      x = M - v[i];
      for (j = i; j <=
           m; j++) v[j] = (v[j] + (ull) x * u[j]) % M;
    }
    ++k;
  }
  for (i =
       k; i < n; i++) if (a[i][m]) return {-1, {}, {}};
  vector<ui> r(m);
  vector<vector<ui>> c;
  for (i = 0;
       i < m; i++) if (fix[i] != -1) r[i]=a[fix[i]][m];
  for (i = 0; i < m; i++) if (fix[i] == -1)
  {
    vector<ui> r(m);
    r[i] = 1;
    for (j = 0; j < m; j++) if
         (fix[j] != -1) r[j] = (M - a[fix[j]][i]) % M;
    c.push_back(r);
  }
  return {R, r, c};
}

/*
usage:
vector<vector<ui>> a(n, vector<ui>(m, 0u));
for (auto &v : a) { // A
  for (auto &x : v) cin >> x;
}
for (auto &v : a) { // B
  int x; cin >> x;
  v.push_back(x);
}
*/
```

7.9 Gaussian Integer gcd

```
typedef complex<ll> cpx;
cpx gaussian_gcd(cpx a, cpx b){
  #define rnd(a
      , b) ((a >= 0 ? a * 2 + b : a * 2 - b) / (b * 2))
  ll real = a.real() * b.real() + a.imag() * b.imag();
  ll imag = a.imag() * b.real() - a.real() * b.imag();
  ll scale = b.real() * b.real() + b.imag() * b.imag();
  if(real % scale == 0 && imag % scale == 0) return b;
  return gaussian_gcd(b, a
      - cpx(rnd(real, scale), (rnd(imag, scale))) * b);
}
```

7.10 Fraction

```
struct fraction {
  ll n, d;
  fraction
      (const ll &_n=0, const ll &_d=1): n(_n), d(_d) {
    ll t = gcd(n, d);
    n /= t, d /= t;
    if (d < 0) n = -n, d = -d;
  }
  fraction operator-() const {
    return fraction(-n, d);
  }
  fraction operator+(const fraction &b) const {
    return fraction(n * b.d + b.n * d, d * b.d);
  }
  fraction operator-(const fraction &b) const {
    return fraction(n * b.d - b.n * d, d * b.d);
  }
  fraction operator*(const fraction &b) const {
    return fraction(n * b.n, d * b.d);
  }
  fraction operator/(const fraction &b) const {
    return fraction(n * b.d, d * b.n);
  }
  void print(){
    cout << n;
    if (d != 1) cout << "/" << d;
  }
};
```

7.11 Miller Rabin

```
// choices of a
// n < 4,759,123,141 3 : 2, 7, 61
// n < 1,122,004,669,633 4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383 6 : primes <= 13
// n < 2^64 7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022

typedef unsigned long long int ull;
using u128 = __uint128_t;

bool Miller_Rabin(ull a, ull n){
  if(n < 2) return 0;
  if((a %= n) == 0) return true;
  if(n % 2 == 0) return n == 2;
  // n - 1 = 2^s * d
  ull d = (n-1) / ((n-1)&(1-n));
  ull s = __lg((n-1)&(1-n));
  ull x = 1;
  // x = a^d
  while(d){
    if(d&1)
      x = (u128)x * a % n;
    d >>= 1;
    a = (u128)a * a % n;
  }
  if(x == 1 || x == n-1) return true;
  while(--s){
    if ((x = (u128)x * x % n) == n - 1) return true;
  }
  return false;
}
```

7.12 Pollard Rho

```
// O(n^(1/4))
ll mul(ll
     a, ll b, ll mod) { return (__int128)a * b % mod; }
bool Miller_Rabin(ll a, ll n) {
  if((a = a % n) == 0) return 1;
  if((n & 1) ^ 1) return n == 2;
  ll tmp = (n - 1) / ((n - 1) & (1 - n));
  ll t = __lg(((n - 1) & (1 - n))), x = 1;
  for(; tmp; tmp >>= 1, a = mul(a, a, n))
    if(tmp & 1) x = mul(x, a, n);
  if(x == 1 || x == n - 1) return 1;
  while(--t)
    if((x = mul(x, x, n)) == n - 1) return 1;
  return 0;
}
// n < 4,759,123,141       3 : 2, 7, 61
// n < 1,122,004,669,633   4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383   6 : pirmes <= 13
// n < 2^64                7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
int prime(ll p) {
    const static int base[7] = {2,
        325, 9375, 28178, 450775, 9780504, 1795265022};
  for (int i = 0; i < 7; ++i)
    if (!Miller_Rabin(base[i], p))
      return 0;
  return 1;
}

map<ll, int> cnt;
void PollardRho(ll n) {
  if (n == 1) return;
  if (prime(n)) return ++cnt[n], void();
  if (n % 2
      == 0) return PollardRho(n / 2), ++cnt[2], void();
  ll x = 2, y = 2, d = 1, p = 1;
  #define f(x, n, p) ((mul(x, x, n) + p) % n)
  while (true) {
    if (d != n && d != 1) {
      PollardRho(n / d);
      PollardRho(d);
      return;
    }
    if (d == n) ++p;
    x = f(x, n, p); y = f(f(y, n, p), n, p);
    d = gcd(abs(x - y), n);
  }
}

// usage:
// PollardRho(x)
// for (auto [num, c] : cnt) x has num^c
```

7.13 Factorial without Prime Factor

```
// O(p^k + log^2 n), pk = p^k
ll prod[MAXP];
ll fac_no_p(ll n, ll p, ll pk) {
  prod[0] = 1;
  for (int i = 1; i <= pk; ++i)
    if (i % p) prod[i] = prod[i - 1] * i % pk;
    else prod[i] = prod[i - 1];
  ll rt = 1;
  for (; n; n /= p) {
    rt = rt * mpow(prod[pk], n / pk, pk) % pk;
    rt = rt * prod[n % pk] % pk;
  }
  return rt;
} // (n! without factor p) % p^k
```

7.14 Discrete Log

```
// Returns minimum
     x for which a ^ x % m = b % m, gcd(a,m) >= 1
int solve(int a, int b, int m) {
    a %= m, b %= m;

    int k = 1, add = 0, g;
    // if a,
        m is definitely co-prime, remove the while loop
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;

    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

7.15 PiCount

```
// return
     pi(n), where pi(n) is the nubmer of primes <= n
ll PrimeCount(ll n) { // n ~ 10^13 => < 2s
  if (n <= 1) return 0;
  int v = sqrt(n), s = (v + 1) / 2, pc = 0;
  vector<int> smalls(v + 1), skip(v + 1), roughs(s);
  vector<ll> larges(s);
  for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
  for (int i = 0; i < s; ++i) {
    roughs[i] = 2 * i + 1;
    larges[i] = (n / (2 * i + 1) + 1) / 2;
  }
  for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
      int q = p * p;
      ++pc;
      if (1LL * q * q > n) break;
      skip[p] = 1;
      for (int i = q; i <= v; i += 2 * p) skip[i] = 1;
      int ns = 0;
      for (int k = 0; k < s; ++k) {
        int i = roughs[k];
        if (skip[i]) continue;
        ll d = 1LL * i * p;
        larges[ns] = larges[k] - (d <= v ? larges
            [smalls[d] - pc] : smalls[n / d]) + pc;
        roughs[ns++] = i;
      }
      s = ns;
      for (int j = v / p; j >= p; --j) {
        int c =
             smalls[j] - pc, e = min(j * p + p, v + 1);
        for (int i = j * p; i < e; ++i) smalls[i] -= c;
      }
    }
  }
  for (int k = 1; k < s; ++k) {
    const ll m = n / roughs[k];
    ll t = larges[k] - (pc + k - 1);
    for (int l = 1; l < k; ++l) {
      int p = roughs[l];
      if (1LL * p * p > m) break;
      t -= smalls[m / p] - (pc + l - 1);
    }
    larges[0] -= t;
  }
  return larges[0];
}
```

7.16 Möbius Function

```
const int mxn = 20000 + 5;
int miu[mxn],  vis[mxn];
vector<int> prime;

void init() {
#define eb emplace_back
  miu[1] = 1;
  for (int i = 2; i < mxn; i++) {
    if (!vis[i]) {
      prime.eb(i);
      miu[i] = -1;
    }
    for (int j =
         0; j < SZ(prime) && i * prime[j] < mxn; j++) {
      vis[i * prime[j]] = 1;
      if (i % prime[j]) miu[i * prime[j]] = -miu[i];
      else miu[i * prime[j]] = 0;
    }
  }
}
```

7.17 Sqrt under Mod

```
// sqrt under mod, if no solution return -1
int Jacobi(int a, int m) {
  int s = 1;
  for (; m > 1; ) {
    a %= m;
    if (a == 0) return 0;
    const int r = __builtin_ctz(a);
    if ((r & 1) && ((m + 2) & 4)) s = -s;
    a >>= r;
    if (a & m & 2) s = -s;
    swap(a, m);
  }
  return s;
}

int QuadraticResidue(int a, int p) {
  if (p == 2) return a & 1;
  const int jc = Jacobi(a, p);
  if (jc == 0) return 0;
  if (jc == -1) return -1;
  int b, d;
  for (; ; ) {
    b = rand() % p;
    d = (1LL * b * b + p - a) % p;
    if (Jacobi(d, p) == -1) break;
  }
  int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
  for (int e = (1LL + p) >> 1; e; e >>= 1) {
    if (e & 1) {
      tmp = (1LL *
          g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
      g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
      g0 = tmp;
    }
    tmp = (1LL
        * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
    f1 = (2LL * f0 * f1) % p;
    f0 = tmp;
  }
  return g0;
}
```

7.18 Sum of Floor

```
// \sum_{i = 0}^{n - 1} floor((a * i + b) / m)
ll sum_of_floor(ll n, ll m, ll a, ll b) {
    ll ans = 0;
    if (a >= m) {
        ans += (n - 1) * n * (a / m) / 2;
        a %= m;
    }
    if (b >= m) {
        ans += n * (b / m);
        b %= m;
    }
    ll y_max
         = (a * n + b) / m, x_max = (y_max * m - b);
    if (y_max == 0) return ans;
    ans += (n - (x_max + a - 1) / a) * y_max;
    ans +=
        sum_of_floor(y_max, a, m, (a - x_max % a) % a);
    return ans;
}
```

8 Tree

8.1 Find Centroid

```
// return (max subtree size, node ID)
int sizeSubT[N];
pair<int, int> Centroid(int u, int pa) {
  sizeSubT[u] = 1;
  pair<int, int> res(INT_MAX, -1);
  int mx = 0;
  for (int v : g[u]) {
    if (v == pa) continue;
    res = min(res, Centroid(v, u));
    sizesubT[u] += sizesubT[v];
    mx = max(mx, sizesubT[v]);
  }
  res =
      min(res, make_pair(max(mx, n - sizesubT[u]), u));
  return res;
}
```

8.2 Centroid Decomposition

```
const int Mlg = __lg(MAX) + 2;

struct edge {
  int to, weight;
  edge(int _to, int _w): to(_to), weight(_w) {}
};

vector<edge> edg[MAX];

struct Cen {
  ll val;
  int p, sz, dep;
  Cen() {}
  Cen(int _p, int _d): val(0), p(_p), sz(0), dep(_d) {}
}
cen[MAX];
ll dis[Mlg][MAX];

bool visit[MAX];
vector<int> v;
int sz[MAX], mx[MAX];
void dfs_sz(int id) {
  visit[id] = 1;
  v.push_back(id);
  sz[id] = 1;
  mx[id] = 0;
  for (edge i: edg[id]) {
    if (!visit[i.to]) {
      dfs_sz(i.to);
      mx[id] = max(mx[id], sz[i.to]);
      sz[id] += sz[i.to];
    }
  }
}
void dfs_dis(int id, int cen_dep, ll weight) {
  dis[cen_dep][id] = weight;
  visit[id] = 1;
  for (edge i: edg[id])
    if (!visit[i.to])
      dfs_dis(i.to, cen_dep, weight + i.weight);
}
void build(int id, int cen_dep, int p) {
  dfs_sz(id);
  int nn = v.size();
  int ccen = -1;
  for (int i: v) {
    if (max(nn - sz[i], mx[i]) * 2 <= nn)
      ccen = i;
    visit[i] = 0;
  }
  dfs_dis(ccen, cen_dep, 0);
  for (int i: v) visit[i] = 0;
  v.clear();
  visit[ccen] = 1;
  cen[ccen] = Cen(p, cen_dep);
  for (edge i: edg[ccen])
    if (!visit[i.to])
      build(i.to, cen_dep + 1, ccen);
}

void add(int id, int d) {
  for (int p = id; p != -1; p = cen[p].p) {
    cen[p].val += dis[cen[p].dep][id] * d;
    cen[p].val -= dis[cen[p].dep - 1][id] * d;
    cen[p].sz += d;
  }
}

ll query(int id) {
  ll ret = 0;
  int pre_sz = 0;
  for (int p = id; p != -1; p = cen[p].p) {
    ret += cen[p].val;
    ret += (cen[p].sz - pre_sz) * dis[cen[p].dep][id];
    pre_sz = cen[p].sz;
  }
  return ret;
}
// edg[u].push_back(edge(v,w));
// edg[v].push_back(edge(u,w));
// memset(visit,0,sizeof(visit));
// build(1,1,-1);
// add(u, d)
// query(u)
```

8.3 Heavy-Light Decomposition

```
struct Heavy_light_Decomposition { // 1-base
  // ulink[i]: the head of i-th path
  int n, ulink[MAXN
      ], deep[MAXN], mxson[MAXN], size[MAXN], pa[MAXN];
  // dt[i]: weight
      on node[i]'s edge to its parent (data of node[i])
  // bln[i]:
      edge[i]'s node's order of dfs (index in seg tree)
  // pl[i]: node[i]'s order of dfs (index in seg tree)
  // data[i]: dt[segtree[i]]
  int t, pl[MAXN], bln[MAXN], edge[MAXN], et;
  vector<int> data, dt;
  vector<pii> G[MAXN];
  void init(int _n) {
    n = _n, t = 0, et = 1;
    for (int i = 1; i <= n; ++i){
      G[i].clear(), mxson[i] = 0;
    }
    data.resize(n+1), dt.resize(n+1);
  }
  void add_edge(int a, int b, int w=0) {
    G[a].push_back(pii(b, et));
    G[b].push_back(pii(a, et));
    edge[et++] = w;
  }
  void dfs(int u, int f, int d) {
    size[u] = 1, pa[u] = f, deep[u] = d++;
    for (auto &i : G[u])
      if (i.X != f) {
        dfs(i.X, u, d), size[u] += size[i.X];
        if (size[mxson[u]] < size[i.X]) mxson[u] = i.X;
      } //else bln[i.Y] = u, dt[u] = edge[i.Y];
  }
  void cut(int u, int link) {
    data[pl[u] = t++] = dt[u], ulink[u] = link;
    if (!mxson[u]) return;
    cut(mxson[u], link);
    for (auto i : G[u])
      if (i.X != pa[u] && i.X != mxson[u])
        cut(i.X, i.X);
  }
  void build(int root) {
    dfs(root, root, 1);
    cut(root, root);
    seg.build(data, n);
  }

  void update(int a, int b, int c){
    c %= mod;
    int ta = ulink[a], tb = ulink[b];
    while(ta != tb){
      if(deep[ta] < deep[tb]){
        seg.update(pl[tb], pl[b], c);
        tb = ulink[b = pa[tb]];
      } else {
        seg.update(pl[ta],pl[a], c);
        ta = ulink[a = pa[ta]];
      }
    }
    if (pl[a] > pl[b]) swap(a, b);
    seg.update(pl[a], pl[b], c);
  }

  int query(int a, int b) {
    int ta = ulink[a], tb = ulink[b];
    int re = 0;
    while (ta != tb){
      if (deep[ta] < deep[tb]) {
        re = re + seg.query(pl[tb], pl[b]);
        tb = ulink[b = pa[tb]];
      } else {
        re = re + seg.query(pl[ta], pl[a]);
        ta = ulink[a = pa[ta]];
      }
    }
    if (pl[a] > pl[b]) swap(a, b);
    re = re + seg.query(pl[a], pl[b]);
    return re;
  }

  void pure_update(int x, int z){
    seg.update(pl[x], pl[x]+size[x]-1, z);
  }

  int pure_query(int x){
    return seg.query(pl[x], pl[x]+size[x]-1);
  }
} tree;
// tree.init(n) -> add_edge(a,b) -> build(root)
// O((logn)^2)
// update(x, y, z) -> modify path from x to y
// query(x, y) -> query path from x to y
// O(logn)
// pure_update(x,z) -> modify x subtree
// pure_query(x) -> query x subtree
```

8.4 LCA

```
const int MAXN = 500010;
const int MAXD = ceil(log2(MAXN));
struct LCA{ //1-base
  vector<int> G[MAXN];
  int n;
  int d[MAXN], lg[MAXN];
  int an[MAXN][MAXD];

  void init(int _n){
    n = _n;
    for(int i = 0; i <= n; i++){
      G[i].clear();
    }
    memset(d,0,sizeof(d));
    memset(lg,0,sizeof(lg));
    memset(an,-1,sizeof(an));
    // precompute
    lg[1] = 0;
    for (int i = 2; i < MAXN; ++i)
      lg[i] = lg[i / 2] + 1;
  }

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

  void dfs(int u, int p = -1, int depth = 0) {
    d[u] = depth;
    an[u][0] = p;
    for (int i
        = 1; i <= lg[n - 1] && an[u][i - 1] != -1; ++i)
      an[u][i] = an[an[u][i - 1]][i - 1];
      // 2^i = 2^(i-1) + 2^(i-1)
    for (auto& v : G[u]) {
      if (v == p) continue; // parent
      dfs(v, u, depth + 1);
    }
  }

  int query(int u, int v) {
    if (d[u] > d[v]) swap(u, v);
    for (int i = lg[d[v] - d[u]]; i >= 0; --i)
      if (d[v] - d[u] >= (1 << i))
        v = an[v][i];
    // v move to same depth
    if (u == v) return u;

    for (int i = lg[d[u]]; i >= 0; --i)
      if (an[u][i] != an[v][i])
        u = an[u][i], v = an[v][i];

    // lca(u,v)
    return an[u][0];
  }
};
// build O(VlogV), query O(logV)
// init -> add_edge -> dfs(root) -> query
LCA tree;
```

8.5 Tree Hash

```
typedef unsigned long long ull;

const ull mask = std::chrono
    ::steady_clock::now().time_since_epoch().count();

ull shift(ull x) {
  x ^= mask;
  x ^= x << 13;
  x ^= x >> 7;
  x ^= x << 17;
  x ^= mask;
  return x;
}

const int N = 1e6 + 10;

ull hash[N];
vector<int> edge[N];

void dfs(int x, int p) {
  hash[x] = 1;
  for (int i : edge[x]) {
    if (i == p) {
      continue;
    }
    dfs(i, x);
    hash[x] += shift(hash[i]);
  }
}
```

```
// usage:
// dfs(1, 0)
```

9 Geometry

9.1 Default Code

```
typedef pair<double,double> pdd;
typedef pair<pdd,pdd> Line;
struct Cir{pdd O; double R;};
const double eps=1e-8;
const double PI = acos(-1.0);
pdd operator+(const pdd &a, const pdd &b)
{ return pdd(a.X + b.X, a.Y + b.Y);}
pdd operator-(const pdd &a, const pdd &b)
{ return pdd(a.X - b.X, a.Y - b.Y);}
pdd operator*(const pdd &a, const double &b)
{ return pdd(a.X * b, a.Y * b);}
pdd operator/(const pdd &a, const double &b)
{ return pdd(a.X / b, a.Y / b);}
double dot(const pdd &a,const pdd &b)
{ return a.X * b.X + a.Y * b.Y;}
double cross(const pdd &a,const pdd &b)
{ return a.X * b.Y - a.Y * b.X;}
double abs2(const pdd &a)
{ return dot(a, a);}
double abs(const pdd &a)
{ return sqrt(dot(a, a));}
int sign(const double &a)
{ return fabs(a) < eps ? 0 : a > 0 ? 1 : -1;}
int ori(const pdd &a,const pdd &b,const pdd &c)
{ return sign(cross(b - a, c - a));}
bool collinearity
    (const pdd &p1, const pdd &p2, const pdd &p3)
{ return sign(cross(p1 - p3, p2 - p3)) == 0;}
bool btw(const pdd &p1,const pdd &p2,const pdd &p3) {
  if(!collinearity(p1, p2, p3)) return 0;
  return sign(dot(p1 - p3, p2 - p3)) <= 0;
}
bool seg_intersect(const pdd
     &p1,const pdd &p2,const pdd &p3,const pdd &p4) {
  int a123 = ori(p1, p2, p3);
  int a124 = ori(p1, p2, p4);
  int a341 = ori(p3, p4, p1);
  int a342 = ori(p3, p4, p2);
  if(a123 == 0 && a124 == 0)
    return btw(p1, p2, p3) || btw(p1, p2, p4) ||
      btw(p3, p4, p1) || btw(p3, p4, p2);
  return a123 * a124 <= 0 && a341 * a342 <= 0;
}
pdd intersect(const pdd
    &p1, const pdd &p2, const pdd &p3, const pdd &p4) {
  double a123 = cross(p2 - p1, p3 - p1);
  double a124 = cross(p2 - p1, p4 - p1);
  return (p4 * a123 - p3 * a124) / (a123 - a124);
}
pdd perp(const pdd &p1)
{ return pdd(-p1.Y, p1.X); }
pdd projection(const pdd &p1, const pdd &p2,
     const pdd &p3) // coordinate of p3 project on p1p2
{ return p1 + (
    p2 - p1) * dot(p3 - p1, p2 - p1) / abs2(p2 - p1); }
```

9.2 Convex Hull

```
// using default code of Geometry
// including +, -, *, eps, cross, sign, ori
void hull(vector<pll> &dots) {
  sort(dots.begin(), dots.end());
  vector<pll> ans(1, dots[0]);
  for (int ct = 0;
       ct < 2; ++ct, reverse(dots.begin(), dots.end()))
    for (int i = 1, t = SZ
        (ans); i < SZ(dots); ans.push_back(dots[i++]))
      while (SZ(ans) > t && ori
          (ans[SZ(ans) - 2], ans.back(), dots[i]) <= 0)
        ans.pop_back();
  ans.pop_back(), ans.swap(dots);
}
```

9.3 Polar Angle Sort

```
// using geometry default code
// sign, cross
bool cmp(pdd a, pdd b) {
  #define is_neg(k) (sign
      (k.Y) < 0 || (sign(k.Y) == 0 && sign(k.X) < 0))
  int A = is_neg(a), B = is_neg(b);
  if (A != B)
    return A < B;
  if (sign(cross(a, b)) == 0)
    return abs2(a) < abs2(b);
  return sign(cross(a, b)) > 0;
}
```

9.4 Intersection of two circles

```
// return if two circles are intersect
// using default code of Geometry
// including Cir, abs2, +, -, /, *
bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
  pdd o1 = a.O, o2 = b.O;
  double r1 =
       a.R, r2 = b.R, d2 = abs2(o1 - o2), d = sqrt(d2);
  if(d < max
      (r1, r2) - min(r1, r2) || d > r1 + r2) return 0;
  pdd u = (o1 + o2) * 0.5
       + (o1 - o2) * ((r2 * r2 - r1 * r1) / (2 * d2));
  double A = sqrt((r1 + r2 + d) *
       (r1 - r2 + d) * (r1 + r2 - d) * (-r1 + r2 + d));
  pdd v
       = pdd(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
  p1 = u + v, p2 = u - v;
  return 1;
}
```

9.5 Intersection of polygon and circle

```
// return intersection area of polygon and circle
// Divides into multiple triangle, and sum up
// using default code of Geometry
// including dot, abs, ori

double _area(pdd pa, pdd pb, double r){
  if(abs(pa)<abs(pb)) swap(pa, pb);
  if(abs(pb)<eps) return 0;
  double S, h, theta;
  double a=abs(pb),b=abs(pa),c=abs(pb-pa);
  double cosB = dot(pb,pb-pa) / a / c, B = acos(cosB);
  double cosC = dot(pa,pb) / a / b, C = acos(cosC);
  if(a > r){
    S = (C/2)*r*r;
    h = a*b*sin(C)/c;
    if (h < r && B
        < PI/2) S -= (acos(h/r)*r*r - h*sqrt(r*r-h*h));
  }
  else if(b > r){
    theta = PI - B - asin(sin(B)/r*a);
    S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
  }
  else S = .5*sin(C)*a*b;
  return S;
}
double area_poly_circle(const
     vector<pdd> poly,const pdd &O,const double r){
  double S=0;
  for(int i=0;i<SZ(poly);++i)
    S+=_area(poly[i]-O,poly[(i+1)%SZ(poly
        )]-O,r)*ori(O,poly[i],poly[(i+1)%SZ(poly)]);
  return fabs(S);
}
```

9.6 Intersection of line and circle

```
// return intersection coordinate
// using default code of Geometry
// including -, /, dot, abs2, cross
vector<pdd> circleLine(pdd c, double r, pdd a, pdd b) {
  pdd p
       = a + (b - a) * dot(c - a, b - a) / abs2(b - a);
  double s = cross
      (b - a, c - a), h2 = r * r - s * s / abs2(b - a);
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  pdd h = (b - a) / abs(b - a) * sqrt(h2);
  return {p - h, p + h};
}
```

9.7 PointSegDist

```
// using default code of Geometry
// including -, sign, abs, dot
double PointSegDist(pdd q0, pdd q1, pdd p) {
  if (sign(abs(q0 - q1)) == 0) return abs(q0 - p);
  if (sign(dot(q1 - q0,
      p - q0)) >= 0 && sign(dot(q0 - q1, p - q1)) >= 0)
    return fabs(cross(q1 - q0, p - q0) / abs(q0 - q1));
  return min(abs(p - q0), abs(p - q1));
}
```

9.8 Rotating SweepLine

```
// using default code of Geometry
// including sign, cross, abs2
bool cmp(pdd a, pdd b){
#define is_neg(k) (
    sign(k.Y) < 0 || (sign(k.Y) == 0 && sign(k.X) < 0))
  int A = is_neg(a), B = is_neg(b);
  if(A!=B){
    return A<B;
  }
  if(sign(cross(A, B))==0){
    return abs2(a) < abs2(b);
  }
  return sign(cross(a, b)) > 0;
}
void rotatingSweepLine(vector<pii> &ps){
  int n = SZ(ps);
  vector<int> id(n), pos(n);
  vectpr<pii> line(n*(n-1));
  int m = 0;
  for(int i=0; i<n; ++i){
    for(int j=0; j<n; ++j){
      if(i!=j){
        line[m++] = pii(i, j);
      }
    }
  }
  sort(line
      .begin(), line.end(), [&](pii a, pii b){return
       cmp(ps[a.Y] - ps[a.X], ps[b.Y] - ps[b.X]);});
  iota(id.begin(), id.end(), 0);
  sort(id.begin(), id.end(), [&](int a, int b){
    if(ps[a].Y !=ps[b].Y)
      return ps[a].Y < ps[b].Y;
    return ps[a] < ps[b];
  });//initial order, since (1, 0) is the smallest
  for(int i=0; i<n; ++i){
    pos[id[i]] = i;
  }
  for(int i=0; i<m; ++i){
    auto l = line[i];
    //do something
    tie(pos[l.X], pos[l.Y], id[pos[l.X]], id[pos[l.Y
        ]]) = make_tuple(pos[l.Y], pos[l.X], l.Y, l.X);
  }
}
```

9.9 Minkowski Sum

```
vector<pll> Minkowski(vector<pll> A, vector<pll> B) {
  hull(A), hull(B);
  vector<pll> C(1, A[0] + B[0]), s1, s2;
  for(int i = 0; i < SZ(A); ++i)
    s1.pb(A[(i + 1) % SZ(A)] - A[i]);
  for(int i = 0; i < SZ(B); i++)
    s2.pb(B[(i + 1) % SZ(B)] - B[i]);
  for(int p1 = 0, p2 = 0; p1 < SZ(A) || p2 < SZ(B);)
    if (p2 >= SZ(B)
         || (p1 < SZ(A) && cross(s1[p1], s2[p2]) >= 0))
      C.pb(C.back() + s1[p1++]);
    else
      C.pb(C.back() + s2[p2++]);
  return hull(C), C;
}
```

9.10 Half Plane Intersection

```
pll area_pair(Line a, Line b)
{ return pll(cross(a.Y
     - a.X, b.X - a.X), cross(a.Y - a.X, b.Y - a.X)); }
bool isin(Line l0, Line l1, Line l2) {
  // Check inter(l1, l2) strictly in l0
  auto [a123, a124] = area_pair(l0, l1);
  if (a123
       - a124 < 0) swap(a123, a124), swap(l1.X, l1.Y);
  auto [b123, b124] = area_pair(l0, l2);
  if (b123 - b124 < 0) swap(l2.X, l2.Y);
  auto [c123, c124] = area_pair(l2, l1);
  if (c123 - c124 < 0) c123 *= -1, c124 *= -1;
  return c123
       * (a123 - a124) < a123 * (c123 - c124); // C^4
}
/* Having solution, check size > 2 */
/* --^-- Line.X --^-- Line.Y --^-- */
vector<Line> halfPlaneInter(vector<Line> arr) {
  sort(ALL(arr), [&](Line a, Line b) -> int {
    if (cmp(a.Y - a.X, b.Y - b.X, 0) != -1)
      return cmp(a.Y - a.X, b.Y - b.X, 0);
    return ori(a.X, a.Y, b.Y) < 0;
  });
  deque<Line> dq(1, arr[0]);
  for (auto p : arr) {
    if (cmp(
        dq.back().Y - dq.back().X, p.Y - p.X, 0) == -1)
      continue;
    while (SZ(dq
        ) >= 2 && !isin(p, dq[SZ(dq) - 2], dq.back()))
      dq.pop_back();
    while (SZ(dq) >= 2 && !isin(p, dq[0], dq[1]))
      dq.pop_front();
    dq.pb(p);
  }
  while (SZ(dq)
       >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq.back()))
    dq.pop_back();
  while (SZ(dq) >= 3 && !isin(dq.back(), dq[0], dq[1]))
    dq.pop_front();
  return vector<Line>(ALL(dq));
}
```

9.11 Polygon Area

```
// using default code of Geometry
// includeing -, cross
double polyArea(vector<pdd> &v) {
  double res = 0.0;
  int n = SZ(v);
  for (int i = 1; i < n - 1;
       i++) res += cross(v[i] - v[0], v[i + 1] - v[0]);
  return res / 2.0;
}
```

9.12 Polygon Union Area

```
// using default code of Geometry
// including -, sign, cross, dot

#define mp make_pair

double seg(pdd &o, pdd &a, pdd &b) {
  if (sign(b
      .X - a.X) == 0) return (o.Y - a.Y) / (b.Y - a.Y);
  return (o.X - a.X) / (b.X - a.X);
}

double polyUnion(vector<vector<pdd>> &v) {
  int n = SZ(v); // number of polygons
  double res = 0.0;
  vector<pair<double, int>> s((n + 10) * 60);
  for (int i = 0; i < n; i++) {
    int sz = SZ(v[i]);
    for (int j = 0; j < sz; j++ ) {
      int m = 0;
      s[m++] = mp(0.0, 0);
      s[m++] = mp(1.0, 0);
      pdd a = v[i][j], b = v[i][(j + 1) % sz];
      for (int k = 0; k < n; k++) {
        if (i != k) {
          int sz2 = SZ(v[k]);
          for (int ii = 0; ii < sz2; ii++) {
            pdd c = v[k][ii], d = v[k][(ii + 1) % sz2];
            int c1 = sign(cross(b - a, c - a));
            int c2 = sign(cross(b - a, d - a));
            if (c1 == c2) {
              if (sign(dot(b - a, d - c))) {
                s[m++] = mp(seg(c, a, b), 1);
                s[m++] = mp(seg(c, a, b), -1);
              }
            } else {
              double s1 = cross(d - c, a - c);
              double s2 = cross(d - c, b - c);
              if (c1 >= 0 && c2
                   < 0) s[m++] = mp(s1 / (s1 - s2), 1);
              else if (c1 < 0 && c2 >=
                   0) s[m++] = mp(s1 / (s1 - s2), -1);
            }
          }
        }
      }
      sort(s, s + m);
      double pre
           = min(max(s[0].X, 0.0), 1.0), sum = 0, now;
      int cov = s[0].Y;
      for (int k = 1; k < m; k++) {
        now = min(max(s[k].X, 0.0), 1.0);
        if (!cov) sum += (now - pre);
        cov += s[k].Y;
        pre = now;
      }
      res += cross(a, b) * sum;
    }
  }
  return res / 2.0;
}
```

10 Flow

10.1 SW_MinCut

```
struct SW_Min_Cut {
    static const int maxn = 500 + 5;
    int edge[maxn][maxn];
    int n;
    int vis[maxn], del[maxn], weight[maxn];

    void Init(int _n)
    {
        memset(edge, 0, sizeof(edge));
        memset(del, 0, sizeof(del));
        n = _n;
    }

    void AddEdge(int u, int v, int w)
    {
        edge[u][v] += w;
        edge[v][u] += w;
    }

    void Search(int &s, int &t)
    {
        memset(vis, 0, sizeof(vis));
        memset(weight, 0, sizeof(weight));
        s = t = -1;
        while (true)
        {
            int mx = -1, cur = 0;
            for (int i = 0; i < n; i++)
            {
                if (!del
                    [i] && !vis[i] and mx < weight[i])
                {
                    cur = i, mx = weight[i];
                }
            }
            if (mx == -1) break;
            vis[cur] = 1;
            s = t, t = cur;
            for (int i = 0; i < n; i++)
            {
                if (!vis[i] &&
                    !del[i]) weight[i] += edge[cur][i];
            }
        }
    }

    int Solve()
    {
        int res = INT_MAX;
        for (int i = 0, x, y; i < n - 1; i++)
        {
            Search(x, y);
            res = min(res, weight[y]);
            del[y] = 1;
            for (int j = 0; j < n; j++)
            {
```

```
                edge[
                    x][j] = (edge[j][x] += edge[y][j]);
            }
        }
        return res;
    }
} graph;
```

10.2 Kuhn Munkres

```
struct KM { // 0-base
  int w[MAXN][MAXN], hl[MAXN], hr[MAXN], slk[MAXN], n;
  int fl[MAXN], fr[MAXN], pre[MAXN], qu[MAXN], ql, qr;
  bool vl[MAXN], vr[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) w[i][j] = -INF;
  }
  void add_edge(int a, int b, int wei) {
    w[a][b] = wei;
  }
  bool Check(int x) {
    if (vl[x] = 1, ~fl[x])
      return vr[qu[qr++] = fl[x]] = 1;
    while (~x) swap(x, fr[fl[x] = pre[x]]);
    return 0;
  }
  void Bfs(int s) {
    fill(slk, slk + n, INF);
    fill(vl, vl + n, 0), fill(vr, vr + n, 0);
    ql = qr = 0, qu[qr++] = s, vr[s] = 1;
    while (1) {
      int d;
      while (ql < qr)
        for (int x = 0, y = qu[ql++]; x < n; ++x)
          if (!vl[x] &&
            slk[x] >= (d = hl[x] + hr[y] - w[x][y]))
            if (pre[x] = y, d) slk[x] = d;
            else if (!Check(x)) return;
      d = INF;
      for (int x = 0; x < n; ++x)
        if (!vl[x] && d > slk[x]) d = slk[x];
      for (int x = 0; x < n; ++x) {
        if (vl[x]) hl[x] += d;
        else slk[x] -= d;
        if (vr[x]) hr[x] -= d;
      }
      for (int x = 0; x < n; ++x)
        if (!vl[x] && !slk[x] && !Check(x)) return;
    }
  }
  int Solve() {
    fill(fl, fl + n, -1), fill(fr, fr + n, -1),
      fill(hr, hr + n, 0);
    for (int i = 0; i < n; ++i)
      hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) Bfs(i);
    int res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
  }
};
// complexity: n^3
// usage
// init -> add_edge -> Solve
// match: (i, fl[i])
```

10.3 Bipartite Graph Matching

```
struct Bipartite_Matching { // 0-base
  int l, r;
  int mp[MAXN], mq[MAXN]; // i -> mp[i], mq[i] -> i
  int dis[MAXN], cur[MAXN];

  vector<int> G[MAXN], G1[MAXN];
  void init(int _l, int _r){
    l = _l, r = _r;
    for(int i = 0; i < l; ++i){
      G[i].clear();
    }
    /* only use in vertex cover
    for(int j = 0; j < r; ++j){
      G1[j].clear();
    }
    */
}
void add_edge(int s, int t){
  G[s].push_back(t);
  G1[t].push_back(s);
}
bool dfs(int u){
  for(int &i = cur[u]; i < SZ(G[u]); ++i){
    int e = G[u][i];
    if(mq[e] == -1
        || (dis[mq[e]] == dis[u] + 1 && dfs(mq[e]))){
      mp[u] = e, mq[e] = u;
      return 1;
    }
  }
  dis[u] = -1;
  return 0;
}
bool bfs(){
  int rt = 0;
  queue<int> q;
  fill_n(dis, l, -1);
  for(int i = 0; i < l; ++i)
    if (mp[i] == -1)
      q.push(i), dis[i] = 0;
  while (!q.empty()){
    int u = q.front();
    q.pop();
    for(int e : G[u]){
      if(mq[e] == -1)
        rt = 1;
      else if (dis[mq[e]] == -1){
        q.push(mq[e]);
        dis[mq[e]] = dis[u] + 1;
      }
    }
  }
  return rt;
}
int matching(){
  int rt = 0;
  fill_n(mp, l, -1);
  fill_n(mq, r, -1);
  while(bfs()){
    fill_n(cur, l, 0);
    for(int i = 0; i < l; ++i){
      if(mp[i] == -1 && dfs(i))
        ++ rt;
    }
  }
  return rt;
}

/* only use for vertex cover
// 0: left, 1: right
int type[2][MAXN];
// 0: never in cover set
// -1: maybe in the cover set
    (point with perfect matching, not visited in dfs)
// 1: must be in the cover set
void dfs2(int u, int right){
  if(type[right][u] != -1) return;
  type[right][u] = 0;
  if(right){
    for(auto v : G1[u]){
      type[!right][v] = 1;
      if(mp[v] != u) dfs2(mp[v], right);
    }
  } else {
    for(auto v : G[u]){
      type[!right][v] = 1;
      if(mq[v] != u) dfs2(mq[v], right);
    }
  }
}

void vertex_cover(){
  //NOTE: make sure to run matching first
  fill_n(type[0], l, -1);
  fill_n(type[1], r, -1);
  // run dfs2 on left
  for(int i = 0; i < l; i++){
    if(mp[i] == -1){
      dfs2(i, 0);
    }
  }
  // run dfs2 on right
    for(int i = 0; i < r; i++){
      if(mq[i] == -1){
        dfs2(i, 1);
      }
    }
  }
  */
};
// O(VE)
// init -> add_edge -> matching
// vertex coverset number = matching number
// vertex coverset -> after matching -> vertex_cover
// independent set number
     = minimum path cover number = V - matching number
// independent set -> after
     vertex_cover (return M) -> independent set is V\M
```

10.4 General Graph Matching

```
const int N = 100006, E = (2e5) * 2;

struct Graph {
  // 1-index
  int to[E], bro[E], head[N], e;
  int lnk[N], vis[N], stp, n;
  int per[N];

  void init(int _n) {
    // remember to set every array to 0
    stp = 0;
    e = 1;
    n = _n;
    for (int i = 1; i <= n; i++)
      head[i] = lnk[i] = vis[i] = 0, per[i] = i;
    // random_shuffle(per+1, per+n+1);
  }

  void add_edge(int u, int v) {
    u = per[u], v = per[v];
    to[e] = v, bro[e] = head[u], head[u] = e++;
    to[e] = u, bro[e] = head[v], head[v] = e++;
  }

  bool dfs(int x) {
    vis[x] = stp;
    for (int i = head[x]; i; i = bro[i]) {
      int v = to[i];
      if (!lnk[v]) {
        lnk[x] = v, lnk[v] = x;
        return true;
      } else if (vis[lnk[v]] < stp) {
        int w = lnk[v];
        lnk[x] = v, lnk[v] = x, lnk[w] = 0;
        if (dfs(w)) {
          return true;
        }
        lnk[w] = v, lnk[v] = w, lnk[x] = 0;
      }
    }
    return false;
  }

  int solve() {
    int ans = 0;
    for (int i = 1; i <= n; i++)
      if (!lnk[i]) {
        stp++;
        ans += dfs(i);
      }
    return ans;
  }
} graph;
```

10.5 Maximum Simple Graph Matching

```
struct GenMatch { // 1-base
  int V, pr[N];
  bool el[N][N], inq[N], inp[N], inb[N];
  int st, ed, nb, bk[N], djs[N], ans;
  void init(int _V) {
    V = _V;
    for (int i = 0; i <= V; ++i) {
      for (int j = 0; j <= V; ++j) el[i][j] = 0;
      pr[i] = bk[i] = djs[i] = 0;
      inq[i] = inp[i] = inb[i] = 0;
    }
  }
  void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
  }
  int lca(int u, int v) {
    fill_n(inp, V + 1, 0);
    while (1)
      if (u = djs[u], inp[u] = true, u == st) break;
      else u = bk[pr[u]];
    while (1)
      if (v = djs[v], inp[v]) return v;
      else v = bk[pr[v]];
    return v;
  }
  void upd(int u) {
    for (int v; djs[u] != nb;) {
      v = pr[u], inb[djs[u]] = inb[djs[v]] = true;
      u = bk[v];
      if (djs[u] != nb) bk[u] = v;
    }
  }
  void blo(int u, int v, queue<int> &qe) {
    nb = lca(u, v), fill_n(inb, V + 1, 0);
    upd(u), upd(v);
    if (djs[u] != nb) bk[u] = v;
    if (djs[v] != nb) bk[v] = u;
    for (int tu = 1; tu <= V; ++tu)
      if (inb[djs[tu]])
        if (djs[tu] = nb, !inq[tu])
          qe.push(tu), inq[tu] = 1;
  }
  void flow() {
    fill_n(inq + 1, V, 0), fill_n(bk + 1, V, 0);
    iota(djs + 1, djs + V + 1, 1);
    queue<int> qe;
    qe.push(st), inq[st] = 1, ed = 0;
    while (!qe.empty()) {
      int u = qe.front();
      qe.pop();
      for (int v = 1; v <= V; ++v)
        if (el[u][v] && djs[u] != djs[v] &&
          pr[u] != v) {
          if ((v == st) ||
            (pr[v] > 0 && bk[pr[v]] > 0)) {
            blo(u, v, qe);
          } else if (!bk[v]) {
            if (bk[v] = u, pr[v] > 0) {
              if (!inq[pr[v]]) qe.push(pr[v]);
            } else {
              return ed = v, void();
            }
          }
        }
    }
  }
  void aug() {
    for (int u = ed, v, w; u > 0;)
      v = bk[u], w = pr[v], pr[v] = u, pr[u] = v,
      u = w;
  }
  int solve() {
    fill_n(pr, V + 1, 0), ans = 0;
    for (int u = 1; u <= V; ++u)
      if (!pr[u])
        if (st = u, flow(), ed > 0) aug(), ++ans;
    return ans;
  }
};
```

10.6 Minimum Weight Matching (Clique version)

```
struct Graph { // 0-base (Perfect Match), n is even
  int n, match[N], onstk[N], stk[N], tp;
  ll edge[N][N], dis[N];
  void init(int _n) {
    n = _n, tp = 0;
    for (int i = 0; i < n; ++i) fill_n(edge[i], n, 0);
  }
  void add_edge(int u, int v, ll w) {
    edge[u][v] = edge[v][u] = w;
  }
  bool SPFA(int u) {
    stk[tp++] = u, onstk[u] = 1;
    for (int v = 0; v < n; ++v)
      if (!onstk[v] && match[u] != v) {
        int m = match[v];
        if (dis[m] >
          dis[u] - edge[v][m] + edge[u][v]) {
          dis[m] = dis[u] - edge[v][m] + edge[u][v];
          onstk[v] = 1, stk[tp++] = v;
          if (onstk[m] || SPFA(m)) return 1;
          --tp, onstk[v] = 0;
        }
      }
    onstk[u] = 0, --tp;
    return 0;
  }
  ll solve() { // find a match
    for (int i = 0; i < n; ++i) match[i] = i ^ 1;
    while (1) {
      int found = 0;
      fill_n(dis, n, 0);
      fill_n(onstk, n, 0);
      for (int i = 0; i < n; ++i)
        if (tp = 0, !onstk[i] && SPFA(i))
          for (found = 1; tp >= 2;) {
            int u = stk[--tp];
            int v = stk[--tp];
            match[u] = v, match[v] = u;
          }
      if (!found) break;
    }
    ll ret = 0;
    for (int i = 0; i < n; ++i)
      ret += edge[i][match[i]];
    return ret >> 1;
  }
};
```

10.7 Dinic

```
const int N = 200 + 5; // number of vertices
const ll INF = (1ll << 60) - 1;
#define pb push_back

struct Dinic { // 0-base
  struct edge {
    int to, rev;
    ll flow, cap;
  };
  vector<edge> G[N];
  int n, s, t, dis[N], cur[N], cnt[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < n + 2; ++i)
      G[i].clear(), cnt[i] = 0;
  }
  void add_edge(int u, int v, ll cap) {
    G[u].pb(edge{v, SZ(G[v]), 0, cap});
    G[v].pb(edge{u, SZ(G[u]) - 1, 0, 0});
  }
  ll dfs(int u, ll cap) {
    if (u == t || !cap) return cap;
    for (int &i = cur[u]; i < SZ(G[u]); ++i) {
      edge &e = G[u][i];
      if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
        ll df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
          e.flow += df, G[e.to][e.rev].flow -= df;
          return df;
        }
      }
    }
    dis[u] = -1;
    return 0;
  }
  bool bfs() {
    fill_n(dis, n + 3, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (edge &e : G[u])
        if (!~dis[e.to] && e.flow != e.cap)
          q.push(e.to), dis[e.to] = dis[u] + 1;
    }
    return dis[t] != -1;
  }
  ll maxflow(int _s, int _t) {
    s = _s, t = _t;
    ll flow = 0, df;
    while (bfs()) {
      fill_n(cur, n + 3, 0);
      while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
  }
} dinic;
```

11 Convolution

11.1 FFT

```
const int MAXN = 2 * 262144;
typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acos(-1);
const cplx I(0, 1);
cplx omega[MAXN + 1];
void pre_fft() {
  for (int i = 0; i <= MAXN; i++) {
    omega[i] = exp(i * 2 * PI / MAXN * I);
  }
}
void fft(int n, cplx a[], bool inv = false) {
  int basic = MAXN/n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    for (int i=0; i<mh; i++) {
      cplx w = omega[inv ? MAXN
           - (i * theta % MAXN) : i * theta % MAXN];
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        cplx x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
    theta = (theta * 2) % MAXN;
  }
  int i = 0;
  for (int j = 1;j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j<i) swap(a[i],a[j]);
  }
  if (inv) {
    for (int i=0; i<n; i++) a[i] /= n;
  }
}
cplx a[MAXN], b[MAXN], c[MAXN];
//how to use :
/*
pre_fft();
fft(n,a);
fft(n,b); // n need to be 2^k
for (int i = 0; i < n; i++) {
  c[i] = a[i] * b[i];
}
fft(n,c,1);
*/
```

11.2 NTT

```
// Remember coefficient are mod P
/*
(mod, root)
(65537, 3)
(23068673, 3)
(998244353, 3)
(1107296257, 10)
(2013265921, 31)
(2885681153, 3)
*/
typedef long long ll;
const int maxn = 65536; // must be power of 2

struct NTT {
  ll mod = 2013265921, root = 31;
  ll omega[maxn + 1];
  void prentt() {
    ll x = fpow(root, (mod - 1) / maxn);
    omega[0] = 1;
    for (int i = 1; i <= maxn; ++i) {
      omega[i] = omega[i - 1] * x % mod;
    }
  }
  void real_init(ll _mod, ll _root) {
    mod = _mod;
    root = _root;
    prentt();
  }
  ll fpow(ll a, ll n) {
    (n += mod - 1) %= mod - 1;
    ll r = 1;
    for (; n; n >>= 1) {
      if (n & 1)(r *= a) %= mod;
      (a *= a) %= mod;
    }
    return r;
  }
  void bitrev(vector<ll> & v, int n) {
    int z = __builtin_ctz(n) - 1;
    for (int i = 0; i < n; ++i) {
      int x = 0;
      for (int j = 0;
           j <= z; ++j) x ^= ((i >> j & 1) << (z - j));
      if (x > i) swap(v[x], v[i]);
    }
  }
  void ntt(vector<ll> & v, int n) {
    bitrev(v, n);
    for (int s = 2; s <= n; s <<= 1) {
      int z = s >> 1;
      for (int i = 0; i < n; i += s) {
        for (int k = 0; k < z; ++k) {
          ll x =
              v[i + k + z] * omega[maxn / s * k] % mod;
          v[i + k + z] = (v[i + k] + mod - x) % mod;
          (v[i + k] += x) %= mod;
        }
      }
    }
  }
  void intt(vector<ll> & v, int n) {
    ntt(v, n);
    reverse(v.begin() + 1, v.end());
    ll inv = fpow(n, mod - 2);
    for (int i = 0; i < n; ++i) {
      (v[i] *= inv) %= mod;
    }
  }
  vector<ll> conv(vector<ll> a, vector<ll> b) {
    int sz = 1;
    while (sz < a.size() + b.size() - 1) sz <<= 1;
    vector<ll> c(sz);
    while (a.size() < sz) a.push_back(0);
    while (b.size() < sz) b.push_back(0);
    ntt(a, sz), ntt(b, sz);
    for (int
        i = 0; i < sz; ++i) c[i] = (a[i] * b[i]) % mod;
    intt(c, sz);
    // len is a.sz + b.sz - 1, no need to pop
    // while (c.size() && c.back() == 0) c.pop_back();
    return c;
  }
};
ll chinese(ll b1, ll m1, ll b2, ll m2) {
  ll a1 = bigpow(m2, m1 - 2, m1) * b1 % m1;
  ll a2 = bigpow(m1, m2 - 2, m2) * b2 % m2;
  ll ret = (a1 * m2 + a2 * m1) % (m1 * m2);
  assert(ret % m1 == b1 && ret % m2 == b2);
  return ret;
}
```

11.3 FWT

```
void FWT(ll a[],int n) {
  for (int
      d = 1; d < n; d <<= 1)  // d = half of block size
    for (int i = 0; i < n; i += d + d) // every block
      for (int j = i; j < i + d; j++) {   //processing
        ll x = a[j], y = a[j + d];
        a[j] = x + y;           //FWT XOR
        a[j + d] = x - y;       //FWT XOR
        a[j] = x + y;           //FWT AND
        a[j + d] = y + x;       //FWT OR

        a[j] = (x + y) / 2;     //IFWT XOR
        a[j + d] = (x - y) / 2; //IFWT XOR
        a[j] = x - y;           //IFWT AND
        a[j + d] = y - x;       //IFWT OR
      }
}
```

12 Else

12.1 Second-Best Minimum Spanning Tree

```
const int MAXN = 1e5 + 10;
const int MAXM = 3e5 + 10;
const int MAXD = ceil(log2(MAXN));
const ll INF = (1ll << 62);

struct Edge {
  int u, v, w;
  // u->v
  bool operator < (const Edge &rhs) const {
    return w < rhs.w;
  }
} E[MAXM];

int N, M, fa[MAXN], vis[MAXM]; //vis: is in MST
vector<pii> v[MAXN];
int an[MAXN
    ][MAXD], mx[MAXN][MAXD], me[MAXN][MAXD], d[MAXN];
ll sum;
int find(int x) {
  return fa[x] == x ? fa[x] : fa[x] = find(fa[x]);
}
void Kruskal() { //1-base
  sort(E + 1, E + M + 1);
  int total = 0;
  for (int i = 1; i <= M; i++) {
    int x =
        E[i].u, y = E[i].v, fx = find(x), fy = find(y);
    if(x == y) continue;
    if (fx != fy) {
      fa[fx] = fy;
      total++, sum += E[i].w, vis[i] = 1;
      v[x].push_back(make_pair(y, E[i].w));
      v[y].push_back(make_pair(x, E[i].w));
    }
    if (total == N - 1) break;
  }
}

void dfs(int x, int p) {
  d[x] = d[p] + 1; an[x][0] = p;
  for(int i = 0, to; i < SZ(v[x]); i++) {
    if((to = v[x][i].X) == p) continue;
    mx[to][0] = v[x][i].Y;
    dfs(to, x);
  }
}

void pre() {
  for (int i = 1; i <= MAXD-1; i++) {
    for (int j = 1; j <= N; j++) {
      an[j][i] = an[an[j][i - 1]][i - 1];
      int topf = an[j][i - 1]; // 2^i-1 ancestor
      mx[j][i] = max(mx[j][i - 1], mx[topf][i - 1]);
      me[j][i] = max(me[j][i - 1], me[topf][i - 1]);
      if (mx[j][i - 1] > mx[topf][i - 1])
           me[j][i] = max(me[j][i], mx[topf][i - 1]);
      else if (mx[j][i - 1] < mx[topf][i
          - 1]) me[j][i] = max(me[j][i], mx[j][i - 1]);
    }
  }
}

int LCA(int x, int y) {
  if (d[x] < d[y]) swap(x, y);
  for (int i = MAXD-1; i
      >= 0; i--) if (d[an[x][i]] >= d[y]) x = an[x][i];
  if (x == y) return x;
  for (int i = MAXD-1; i >= 0; i--) if (
      an[x][i] != an[y][i]) x = an[x][i], y = an[y][i];
  return an[x][0];
}

int findmax(int x, int lca, int val) {
  ll ans = 0;
  for (int i = MAXD-1; i >= 0; i--) {
    if (d[an[x][i]] >= d[lca]) {
      // if (mx
          [x][i] == val) ans = max(ans, (ll) me[x][i]);
      // else
      ans = max(ans, (ll) mx[x][i]);
      x = an[x][i];
    }
  }
  return ans;
}

void work() {
  ll ans = INF;
  for (int i = 1; i <= M; i++) {
    if (vis[i]) continue;
    int x = E[i].u, y = E[i].v, z = E[i].w;
    if(x == y) continue;
    int lca = LCA(x, y);
    int lmx
        = findmax(x, lca, z), rmx = findmax(y, lca, z);
    // if (max(lmx, rmx) != z)
    ans = min(ans, sum + z - max(lmx, rmx));
  }
  cout << ans << endl;
}

void init(){
  for(int i = 1; i <= N; i++){
    fa[i] = i;
    v[i].clear();
  }
  // memset(E, 0, sizeof(E));
  memset(vis, 0, sizeof(vis));
  memset(an, 0, sizeof(an));
  memset(mx, 0, sizeof(mx));
  memset(me, 0, sizeof(me));
  memset(d, 0, sizeof(d));
  sum = 0;
}

void add_edge(int i, int x, int y, int z){
  E[i] = (Edge) {x, y, z};
}

void solver(){
  Kruskal();
  dfs(1, 0);
  pre();
  work();
}
// O(M log N)
// init -> add_edge -> solver
```

12.2 Algorithm Note

```
1. articulation point & bridge
  if deleted, then connected component ++
2. tree centroid
  if deleted, then T will be separated
      into multiple trees, where nodes of each <= n / 2
3. min cut:
   - a cut
        such that all nodes can be separated into two
        sets S, T = V - S, and the capacity is minimum
   - how to find
     do flow
     block cap = 0 edge
     bfs from s, can reach -> S, can't reach -> T
4. bipartite matching using dinic
  - after doing dinic.maxflow
  - for (int i = 1; i <= m; i++) {
      for (auto [to, rev, flow, cap] : dinic.G[i]) {
        if (flow == 1) cout << i << " " << to << "\n";
      }
    }
```

13 Python

13.1 Misc

```
from decimal import *
setcontext(Context(prec
    =MAX_PREC, Emax=MAX_EMAX, rounding=ROUND_FLOOR))
print(Decimal(input()) * Decimal(input()))
from fractions import Fraction
Fraction('3.1415926535897932').limit_denominator(1000)
```