ToC

1 General

1.1 vimrc

```
"place at ~./vimrc"
sy on
set ru nu rnu cul cin et
    bs=2 ls=2 so=8 sw=4 sts=4 mouse=a
inoremap ( ()<Esc>i
inoremap [ []<Esc>i
inoremap " ""<Esc>i
inoremap ' ''<Esc>i
inoremap {<CR> {<CR>}<Esc>O
noremap <F9> <Esc>:w<CR>:!g++ "%:p" -o "%:p:r".out
    -std=c++14 -DNYAOWO -O2 -Wall -Wextra -Wshadow
    -Wconversion -fsanitize=address -fsanitize=
    undefined -D_GLIBCXX_DEBUG<CR>
noremap <F10> <Esc>:!"%:p:r".out<CR>
map <F11> <F9><F10>
```

1.2 Default Code

```
#include <bits/stdc++.h>
#define int int64_t
#define double long double
using namespace std;
using i128 = __int128_t;
using pii = pair<int, int>;
#define For(i, a, b) for (int i = a; i <= b; i++)
#define Forr(i, a, b) for (int i = a; i >= b; i--)
#define F first
#define S second
#define eb emplace_back
#define all(x) x.begin(), x.end()
#define sz(x) ((int)x.size())
#define INF (int)(4e18)

int32_t main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
}
```

1.3 Splitmix64

```
const uint64_t FIXED = 1 |
  chrono::steady_clock::now()
    .time_since_epoch()
    .count();
struct MyHash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  uint64_t operator()(uint64_t x) const {
    return splitmix64(x + FIXED);
  }
  uint64_t operator()(const pii &p) const {
    return splitmix64(p.F * FIXED + p.S);
  }
};
```

1.4 Optimization

```
int mulll(int a, int b, int M) {
  int r = a * b - M * int(1.L / M * a * b);
  return r + M * ((r < 0) - (r >= (int)M));
}
```

2 Math & Polynomials

2.1 Theorems

- Pick's Theorem

  if a polygon has integer coords for all vertices, then

  $$A = I + \frac{B}{2} - 1$$

  , where A = area, I = internal points, B = points on boundary.

2.2 Fast pow

```
int fpow(int b, int p, const int &mod = MOD) {
  int ans = 1;
  for (; p; p >>= 1, b = b * b % mod)
    if (p & 1) ans = ans * b % mod;
  return ans;
}
```

2.3 Extended GCD

```
int exgcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int g = exgcd(b, a % b, y, x);
  y -= x * (a / b);
  return g;
}
```

2.4 Prime Sieve

```
vector<int> p;
bitset<MAXN + 10> notp;
void build() {
  For(i, 2, MAXN) {
    if (!notp[i]) p.eb(i);
    for (auto &j : p) {
      if (i * j > MAXN) break;
      notp[i * j] = 1;
      if (i % j == 0) break;
    }
  }
}
```

2.5 Prime Numbers

```
12,721 / 13,331 / 123,457 / 222,557 / 999,983
100,102,021 / 1,000,000,021 / 1,000,512,343
1,000,000,000,039 / 1,000,000,000,000,037
2,305,843,009,213,693,951
4,611,686,018,427,387,847
9,223,372,036,854,775,783
18,446,744,073,709,551,557
```

2.6 Mod Inverse

```
int rev[MAXN + 10];
void build() {
  rev[1] = 1;
  For(i, 2, MAXN) {
    rev[i] = rev[MOD % i] * (MOD - MOD / i) % MOD;
  }
}
```

2.7 Chinese Remainder Theorem

```
// avoid __int128 would be WAY FASTER!
struct CRT {
  int k;
  i128 mod;
  vector<i128> p;
  vector<i128> x;
  vector<vector<i128>> pr;
  void init(i128 _mod, vector<i128> &_p) {
    mod = _mod;
    p = _p;
    k = sz(p);
    x.resize(k);
    pr.resize(k);
    For(i, 0, k - 1) {
      pr[i].resize(i);
      For(j, 0, i - 1) pr[i][j] =
        fpow128(p[j], p[i] - 2, p[i]);
    }
  }
  i128 solve(const vector<i128> &r) {
    assert(sz(r) == k);
    For(i, 0, k - 1) {
      x[i] = r[i];
      For(j, 0, i - 1) {
        x[i] = pr[i][j] * (x[i] - x[j]) % p[i];
      }
      if (x[i] < 0) x[i] += p[i];
    }
    i128 ans = 0;
    Forr(i, k - 1, 0) ans = (ans * p[i] + x[i]) % mod;
    return ans;
  }
} crt;
```

2.8 Millar Robin

```
bool is_prime(int n) {
  if (n <= 3 or ((n % 6) & 3) != 1)
    return (n | 1) == 3;
  const int owo[] = {
    2, 325, 9375, 28178, 450775, 9780504, 1795265022};
  int r = __builtin_ctzll((int64_t)(n - 1));
  int d = n >> r;
  for (auto &p : owo)
    if (p % n) {
      int x = fpowll(p % n, d, n), i = r;
      while (x != 1 && x != n - 1 && i--)
        x = mulll(x, x, n);
      if (i != r && x != n - 1) return false;
    }
  return true;
}
```

2.9 Pollard's Rho

```
int rho(int n) {
  static int c = 48763;
  auto next = [&n](int x) {
    return (mulll(x, x, n) + c);
  };
  int a = 2, b = 2, g, gg = 1;
  for (int r = 1; (r & 127) || __gcd(gg, n) == 1;
       r++) {
    if (a == b) {
      c = rng() % (n - 1) + 1;
      a = 2;
      b = next(2);
    }
    g = mulll(gg, (a > b ? a - b : b - a), n);
    gg = g ? g : gg;
    a = next(a);
    b = next(next(b));
  }
  return __gcd(gg, n);
}
```

2.10 Polynomials

2.10.1 FFT*

```
using cpx = complex<double>;
const double PI = acos(-1);
void FFT(vector<cpx> &v, int n, bool rev) {
  assert(__builtin_popcountll(n) == 1);
  for (int i = 0, j = 0; i < n; i++) {
    if (i < j) swap(v[i], v[j]);
    for (int k = n >> 1; (j ^= k) < k; k >>= 1)
      ;
  }
  for (int m = 1; m < n; m <<= 1) {
    cpx omega =
      exp(cpx(0, (rev ? -1 : 1) * 2 * PI / (m * 2)));
    for (int s = 0; s < n; s += m * 2) {
      cpx now = 1;
      for (int i = 0; i < m; i++) {
        cpx u = v[s + i];
        cpx t = v[s + i + m] * now;
        v[s + i] = u + t;
        v[s + i + m] = u - t;
        now *= omega;
      }
    }
  }
  if (rev)
    for (auto &i : v) i /= n;
}
vector<cpx> convolution(vector<cpx> a, vector<cpx> b) {
  int n = 1 << (__lg(sz(a) + sz(b)) + 1);
```

```
  a.resize(n);
  b.resize(n);
  FFT(a, n, 0);
  FFT(b, n, 0);
  vector<cpx> c(n);
  For(i, 0, n - 1) c[i] = a[i] * b[i];
  FFT(c, n, 1);
  return c;
}
```

2.10.2 NTT

```
// avoid __int128 would be WAY FASTER!
void NTT(vector<i128> &a, int n, i128 mod, i128 rt,
  bool rev) {
  for (int i = 0, t = 0; i < n; i++) {
    if (i < t) swap(a[i], a[t]);
    for (int k = n >> 1; (t ^= k) < k; k >>= 1)
      ;
  }
  for (int len = 2; len <= n; len <<= 1) {
    int mi = len >> 1;
    i128 omega = fpow128(rt, (mod - 1) / len, mod);
    if (rev) omega = fpow128(omega, mod - 2, mod);
    Fors(i, 0, n - 1, len) {
      i128 noww = 1;
      For(j, 0, mi - 1) {
        i128 t = a[i + j];
        i128 u = a[i + j + mi] * noww % mod;
        a[i + j] = t + u;
        if (a[i + j] >= mod) a[i + j] -= mod;
        a[i + j + mi] = t - u;
        if (a[i + j + mi] < 0) a[i + j + mi] += mod;
        noww = noww * omega % mod;
      }
    }
  }
  if (rev) {
    i128 n1 = fpow128(n, mod - 2, mod);
    for (auto &i : a) i = i * n1 % mod;
  }
}
vector<i128> convolution(
  vector<i128> a, vector<i128> b, i128 mod, i128 rt) {
  int n = 1 << (__lg((LL)(sz(a) + sz(b))) + 1);
  a.resize(n);
  NTT(a, n, mod, rt, false);
  b.resize(n);
  NTT(b, n, mod, rt, false);
  vector<i128> c(n);
  For(i, 0, n - 1) c[i] = a[i] * b[i] % mod;
  NTT(c, n, mod, rt, true);
  return c;
}
// useful primes: p = r 2^k + 1
//                   p             r     k   root
//            23068673            11    21      3
//           998244353           119    23      3
// 9223372036737335297   54975513881    24      3
//           167772161             5    25      3
//          1107296257            33    25     10
//           469762049             7    26      3
//          2013265921            15    27     31
```

3 Geometry

3.1 Vector Operations

```
struct PT {
  int x, y;
  PT() {}
  PT(int _x, int _y) : x(_x), y(_y) {}
};
PT operator+(const PT &p1, const PT &p2) {
  return PT(p1.x + p2.x, p1.y + p2.y);
}
PT operator-(const PT &p1, const PT &p2) {
  return PT(p1.x - p2.x, p1.y - p2.y);
}
int operator*(const PT &p1, const PT &p2) {
  return p1.x * p2.x + p1.y * p2.y;
}
int operator^(const PT &p1, const PT &p2) {
  return p1.x * p2.y - p1.y * p2.x;
}
PT perp(const PT &p) { return PT(-p.y, p.x); }
int sign(const int &x) {
  return x == 0 ? 0 : (x < 0 ? -1 : 1);
}
int abs2(const PT &x) { return x * x; }
double abs(const PT &x) { return sqrt(abs2(x)); }
int ori(PT p1, PT p2, PT p3) {
  return sign((p2 - p1) ^ (p3 - p1));
}
bool coline(PT p1, PT p2, PT p3) {
  return sign(ori(p1, p2, p3)) == 0;
}
bool btw(PT p1, PT p2, PT p3) {
  if (!coline(p1, p2, p3)) return false;
  return sign((p1 - p3) * (p2 - p3)) <= 0;
}
bool seg_inter(PT p1, PT p2, PT p3, PT p4) {
  int r123 = ori(p1, p2, p3);
  int r124 = ori(p1, p2, p4);
  int r341 = ori(p3, p4, p1);
  int r342 = ori(p3, p4, p2);
  if (r123 == 0 && r124 == 0) {
    return btw(p1, p2, p3) || btw(p1, p2, p4) ||
      btw(p3, p4, p1) || btw(p3, p4, p2);
  }
  return r123 * r124 <= 0 && r341 * r342 <= 0;
}
```

3.2 Convex Hull

```
vector<PT> convex(vector<PT> v) {
  sort(all(v), [&](const PT &a, const PT &b) {
    if (a.x != b.x) return a.x < b.x;
    return a.y < b.y;
  });
  vector<PT> hull;
  For(phase, 0, 1) {
    int t = sz(hull);
    for (auto &p : v) {
      int s = sz(hull);
      while (s - t >= 2 &&
        ori(hull[s - 2], hull[s - 1], p) <= 0) {
        hull.pop_back();
        s--;
      }
      hull.emplace_back(p);
    }
    hull.pop_back();
    reverse(all(v));
  }
  return hull;
}
```

3.3 Polar Angle Sorting

```
bool upperhalf(const PT &p) {
  return p.y > 0 || (p.y == 0 && p.x >= 0);
}
bool pollarCmp(const PT &p1, const PT &p2) {
```

```
  auto u1 = upperhalf(p1);
  auto u2 = upperhalf(p2);
  if (u1 != u2) return u1;
  auto cr = ori(PT(0, 0), p1, p2);
  if (cr != 0) return cr > 0;
  return abs2(p1) < abs2(p2);
}
```

4 Data Structure

4.1 BIT

```
#define LO(x) ((x) & (-x))
struct BIT {
  int n;
  int a[MAXN];
  void init(int _n) {
    n = _n;
    memset(a, 0, sizeof(a));
  }
  void add(int i, int x) {
    while (i <= n) {
      a[i] += x;
      i += LO(i);
    }
  }
  int ask(int i) {
    int ans = 0;
    while (i > 0) {
      ans += a[i];
      i -= LO(i);
    }
    return ans;
  }
} bit;
```

4.2 Segment Tree with Lazy Tags

```
#define L(id) ((id)*2 + 1)
#define R(id) ((id)*2 + 2)
struct SegTree {
  struct SegNode {
    // info & tags
    void tag(...) {
      // put new tags & update info
    }
  } a[MAXN << 2];
  void pull(int id) {
    // merge info
  }
  void push(int id) {
    // push tags
  }
  void build(int id, int l, int r, ...) {
    // init info
    if (l == r) {
      // init info
      return;
    }
    int m = (l + r) / 2;
    build(L(id), l, m, ...);
    build(R(id), m + 1, r, ...);
    pull(id);
  }
  void upd(int id, int l, int r, int L, int R, ...) {
    if (l >= L && r <= R) {
      // update tags & info
      return;
    }
    int m = (l + r) / 2;
    push(id);
    if (L <= m) upd(L(id), l, m, L, R, ...);
    if (R > m) upd(R(id), m + 1, r, L, R, ...);
    pull(id);
  }
  int ask(int id, int l, int r, int L, int R, ...) {
    if (l >= L && r <= R) {
      // get info
    }
    int m = (l + r) / 2;
    push(id);
    int ans = 0;
    if (L <=
      m) // update ans = ask(L(id), l, m, L, R, ...);
      if (R > m) // update ans = ask(R(id), m + 1, r,
                 // L, R, ...);
        pull(id);
    // return ans
  }
} seg;
```

4.3 Treap

```
// implicit key treap
// range reversal + range sum
struct Treap {
  Treap *l, *r;
  int pri, size;
  int val, sum, rev;
  Treap(int _v)
    : l(nullptr), r(nullptr), pri(rng()), size(1),
      val(_v), sum(_v), rev(0) {}
};
int size(Treap *&rt) { return rt ? rt->size : 0; }
int sum(Treap *&rt) { return rt ? rt->sum : 0; }
void pull(Treap *&rt) {
  rt->size = 1 + size(rt->l) + size(rt->r);
  rt->sum = rt->val + sum(rt->l) + sum(rt->r);
}
void push(Treap *&rt) {
  if (rt->rev) {
    rt->rev = 0;
    swap(rt->l, rt->r);
    if (rt->l) rt->l->rev ^= 1;
    if (rt->r) rt->r->rev ^= 1;
  }
}
void split_size(
  Treap *rt, Treap *&a, Treap *&b, int k) {
  if (!rt) {
    a = b = nullptr;
    return;
  }
  push(rt);
  if (size(rt->l) >= k) {
    b = rt;
    split_size(rt->l, a, b->l, k);
    pull(b);
  } else {
    a = rt;
    split_size(rt->r, a->r, b, k - 1 - size(rt->l));
    pull(a);
  }
}
Treap *merge(Treap *a, Treap *b) {
  if (!a || !b) return a ? a : b;
  if (a->pri > b->pri) {
    push(a);
    a->r = merge(a->r, b);
    pull(a);
    return a;
  } else {
    push(b);
    b->l = merge(a, b->l);
    pull(b);
    return b;
```

```
  }
}
```

5 Graph & Flow

5.1 Edge BCC

```
int d[MAXN];
int lo[MAXN];
vector<vector<int>> ebcc;
void getEBCC(int n, int p = 1, int dep = 1) {
  if (d[n]) return;
  static vector<int> st;
  d[n] = lo[n] = dep;
  int visp = 0;
  st.eb(n);
  for (auto &i : adj[n]) {
    if (i != n) {
      if (!visp && i == p) visp = 1;
      else if (d[i] != 0) lo[n] = min(lo[n], d[i]);
      else {
        dfs_ebcc(i, n, dep + 1);
        lo[n] = min(lo[n], lo[i]);
      }
    }
  }
  if (lo[n] == d[n]) { // edge BCC
    if (n != p) ans.eb(p, n); // !!!!
    ebcc.eb();
    do {
      ebcc.back().eb(st.back());
      st.pop_back();
    } while (ebcc.back().back() != n);
  }
}
```

5.2 Articulation Points

```
int d[MAXN];
int lo[MAXN];
bitset<MAXN> isap;
vector<vector<int>> vbcc;
void getVBCC(int n, int p, int dep = 1) {
  if (d[n]) return;
  static vector<int> st;
  d[n] = lo[n] = dep;
  int visp = 0, nc = 0;
  st.eb(n);
  for (auto &i : adj[n])
    if (i != n) {
      if (!visp && i == p) visp = 1;
      else if (d[i] != 0) lo[n] = min(lo[n], d[i]);
      else {
        nc++;
        dfs_vbcc(i, n, dep + 1);
        lo[n] = min(lo[n], lo[i]);
        if (n != p && lo[i] >= d[n]) isap[n] = 1;
        if (lo[i] >= d[n]) { // vertex BCC
          vbcc.eb();
          do {
            vbcc.back().eb(st.back());
            st.pop_back();
          } while (vbcc.back().back() != i);
          vbcc.back().eb(n);
        }
      }
    }
  if (n == p && nc > 1) isap[n] = 1;
}
```

5.3 SCC

```
int sccid[MAXN];
vector<vector<int>> scc;
void getSCC(int n) {
  memset(sccid, 0, sizeof(sccid));
  vector<int> st;
  auto dfs1 = [&](auto dfs, int cur) -> void {
    sccid[cur] = -1;
    for (auto &i : adj[cur]) {
      if (sccid[i] == 0) dfs(dfs, i);
    }
    st.eb(cur);
  };
  auto dfs2 = [&](auto dfs, int cur) -> void {
    scc.back().eb(cur);
    sccid[cur] = sz(scc) - 1;
    for (auto &i : rev[cur]) {
      if (sccid[i] == -1) dfs(dfs, i);
    }
  };
  For(i, 1, n) if (sccid[i] == 0) dfs1(dfs1, i);
  while (sz(st)) {
    if (sccid[st.back()] == -1) {
      scc.eb();
      dfs2(dfs2, st.back());
    }
    while (sz(st) && sccid[st.back()] != -1)
      st.pop_back();
  }
}
```

5.4 Dinic

```
struct Dinic {
  struct Edge {
    int to, rev, cap;
  };
  vector<Edge> g[MAXN];
  int d[MAXN], now[MAXN];
  int s, t;
  void init(int n) { For(i, 0, n - 1) g[i].clear(); }
  void link(int a, int b, int c) {
    g[a].push_back({b, sz(g[b]), c});
    g[b].push_back({a, sz(g[a]) - 1, 0});
  }
  bool bfs() {
    memset(d, -1, sizeof(d));
    d[s] = 0;
    queue<int> que;
    que.emplace(s);
    while (!que.empty()) {
      int k = que.front();
      que.pop();
      for (auto &e : g[k]) {
        if (d[e.to] != -1 || e.cap <= 0) continue;
        d[e.to] = d[k] + 1;
        que.emplace(e.to);
      }
    }
    return d[t] != -1;
  }
  int dfs(int k, int flow) {
    if (flow == 0 || k == t) return flow;
    for (int &i = now[k]; i < sz(g[k]); i++) {
      auto &e = g[k][i];
      if (d[e.to] != d[k] + 1 || e.cap <= 0) continue;
      int f = dfs(e.to, min(flow, e.cap));
      if (f) {
        e.cap -= f;
        g[e.to][e.rev].cap += f;
        return f;
      }
    }
```

```
    d[k] = -1;
    return 0;
  }
  int maxFlow(int _s, int _t) {
    s = _s;
    t = _t;
    int flow = 0, f;
    while (bfs()) {
      memset(now, 0, sizeof(now));
      while ((f = dfs(s, INF)) != 0) flow += f;
    }
    return flow;
  }
} flow;
```

5.5 Min Cost Max Flow

```
struct MCMF {
  struct Edge {
    int to, rev, cap, cost;
  };
  vector<Edge> g[MAXN];
  int dis[MAXN];
  int par[MAXN];
  int pid[MAXN];
  bool inq[MAXN];
  int n;

  void init(int _n) {
    n = _n;
    For(i, 0, n) g[i].clear();
  }
  void link(int u, int v, int cap, int cost) {
    g[u].push_back({v, sz(g[v]), cap, cost});
    g[v].push_back({u, sz(g[u]) - 1, 0, -cost});
  }
  pii maxFlow(int s, int t) {
    int flow = 0, cost = 0;
    while (true) {
      For(i, 0, n) {
        dis[i] = INF;
        inq[i] = false;
      }
      queue<int> que;
      que.push(s);
      dis[s] = 0;
      while (!que.empty()) {
        int now = que.front();
        que.pop();
        inq[now] = false;
        For(i, 0, sz(g[now]) - 1) {
          auto &e = g[now][i];
          if (e.cap > 0 &&
            e.cost + dis[now] < dis[e.to]) {
            dis[e.to] = dis[now] + e.cost;
            par[e.to] = now;
            pid[e.to] = i;
            if (!inq[e.to]) {
              inq[e.to] = true;
              que.push(e.to);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      int mn = INF;
      for (int i = t; i != s; i = par[i]) {
        mn = min(mn, g[par[i]][pid[i]].cap);
      }
      flow += mn;
      cost += mn * dis[t];
      for (int i = t; i != s; i = par[i]) {
        g[par[i]][pid[i]].cap -= mn;
        g[i][g[par[i]][pid[i]].rev].cap += mn;
      }
    }
    return pii(flow, cost);
  }
} flow;
```

6 String

6.1 Z Algorithm

```
void getZ(int n, const char *s, int *z) {
  z[0] = n;
  int l, r;
  l = r = 0;
  For(i, 1, n - 1) {
    if (i < r) z[i] = min(r - i, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]])
      z[i]++;
    if (i + z[i] > r) {
      l = i;
      r = i + z[i];
    }
  }
}
```

6.2 KMP

```
void getKMP(const int n, const char *s, int *f) {
  f[0] = -1;
  int j = -1;
  For(i, 1, n - 1) {
    while (j != -1 && s[j + 1] != s[i]) j = f[j];
    if (s[j + 1] == s[i]) j++;
    f[i] = j;
  }
}
```

6.3 Manacher's Algorithm

```
// do "abc" -> "~a~b~c~" before call
void getManacher(const int n, const char *s, int *m) {
  m[0] = 1;
  int c, l;
  c = l = 0;
  For(i, 1, n - 1) {
    if (i < c + l) m[i] = min(c + l - i, m[c * 2 - i]);
    while (i + m[i] < n && i - m[i] >= 0 &&
      s[i + m[i]] == s[i - m[i]])
      m[i]++;
    if (i + m[i] > c + l) {
      c = i;
      l = m[i];
    }
  }
}
```

6.4 Minimal Rotation

```
int min_rot(int n, string s) {
  s = s + s;
  int i = 0, ans = 0;
  while (i < n) {
    ans = i;
    int j = i + 1, k = i;
    while (j < sz(s) && s[j] >= s[k]) {
      if (s[j] == s[k]) k++;
      else k = i;
      j++;
    }
    while (i <= k) i += j - k;
  }
  return ans;
}
```

6.5 SA & LCP

```cpp
int sa[MAXN + 10];
int rk[MAXN + 10];
int lcp[MAXN + 10];
// build SA in O(N log^2 N)
void getSA(int n, const char *s) {
  vector<int> r(n + 1), r2(n + 1);
  r[n] = r2[n] = -1;
  For(i, 0, n - 1) {
    sa[i] = i;
    r[i] = s[i];
  }
  for (int len = 1; len <= n; len <<= 1) {
    auto cmp = [&](int i, int j) {
      if (r[i] != r[j]) return r[i] < r[j];
      i += len;
      j += len;
      if (i < n && j < n) return r[i] < r[j];
      return i > j;
    };
    sort(sa, sa + n, cmp);
    r2[sa[0]] = 0;
    For(i, 1, n - 1) r2[sa[i]] =
      r2[sa[i - 1]] + cmp(sa[i - 1], sa[i]);
    swap(r, r2);
    if (r[n - 1] == n - 1) return;
  }
}
// call after getSA()
void getLCP(int n, const char *s) {
  For(i, 0, n - 1) rk[sa[i]] = i;
  for (int now = 0, i = 0; i < n; i++) {
    if (rk[i] == 0) lcp[rk[i]] = 0;
    else {
      if (now) now--;
      int j = sa[rk[i] - 1];
      while (i + now < n && j + now < n &&
        s[i + now] == s[j + now])
        now++;
      lcp[rk[i]] = now;
    }
  }
}
```