Contents

1 Miscellaneous

1.1 What to Look for

```
Wrong answer:
Print your solution! (Print debug output, as well.)
Are you clearing all DS between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables? Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Go for a small walk, e.g. to the toilet. Is your output format correct? (including ↩
    whitespace)
Recode

Runtime error: In some judges, MLE is considered RTE (Check your memory use)
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Infinite recursion?
Any possible division by 0? Ex. mod 0
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits.
Force WA (asserts for RTE problems)

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all DS between test cases?
Infinite recursion? (As in pushing into a DS infinitely)
```

1.2 Day of Date

```cpp
// 0-based
const vector<int> T = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4}
int day(int d, int m, int y) {
  y -= (m < 3);
  return (y + y / 4 - y / 100 + y / 400 + T[m - 1] + d) % 7;
}
```

1.3 Number of Days since 1-1-1

```cpp
int rdn(int d, int m, int y) {
  if(m < 3)
    --y, m += 12;
  return 365 * y + y / 4 - y / 100 + y / 400
        + (153 * m - 457) / 5 + d - 306;
}
```

Owen Djonatan, Luis Anthonie Alkins, Richard Alison
Ownicron, ZeroScar, KerakTelor

Proof by... forgor
Bina Nusantara University

Page 1 of 25

1.4 Enumerate Subsets of a Bitmask

```
int x = 0;
do {
  // do stuff with the bitmask here
  x = (x + 1 + ~m) & m;
} while(x != 0);
```

1.5 Fast IO

```
int read() {
  char c;
  do {
    c = getchar_unlocked();
  } while(c < 33);
  int res = 0, mul = 1;
  if(c == '-') {
    mul = -1;
    c = getchar_unlocked();
  }
  for(; '0' <= c && c <= '9'; c = getchar_unlocked())
    res = res * 10 + c - '0';
  return res * mul;
}

void write(int x) {
  static char wbuf[10];
  if(x < 0) {
    putchar_unlocked('-');
    x = -x;
  }
  int idx = 0;
  for(; x; x /= 10)
    wbuf[idx++] = x % 10;
  if(idx == 0) putchar_unlocked('0');
  for(int i = idx - 1; i >= 0; --i) putchar_unlocked(wbuf[i] + '0');
}
void write(const char* s) {
  while(*s) {
    putchar_unlocked(*s);
    ++s;
  }
}
```

1.6 Josephus Problem

```
ll josephus(ll n, ll k) { // O(k log n)
  if(n == 1) return 0;
  if(k == 1) return n - 1;
  if(k > n) return (josephus(n - 1, k) + k) % n;
  ll cnt = n / k;
  ll res = josephus(n - cnt, k);
  res -= n % k;
  if(res < 0) res += n;
  else res += res / (k - 1);
  return res;
}
int josephus(int n, int k) { // O(n)
  int res = 0;
  for(int i = 1; i <= n; ++i) res = (res + k) % i;
  return res + 1;
}
```

1.7 Random Primes

```
36671 74101 724729 825827 924997 1500005681 2010408371 2010405347
```

1.8 RNG

```
// RNG - rand_int(min, max), inclusive

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

template<class T>
T rand_int(T mn, T mx) {
  return uniform_int_distribution<T>(mn, mx)(rng);
}
```

2 Data Structures

2.1 2D Segment Tree

```
struct Segtree2D {
  struct Segtree {
    struct node {
      int l, r, val;
      node* lc, * rc;
      node(int _l, int _r, int _val = INF) : l(_l), r(_r), val(_val),
        lc(NULL), rc(NULL) {}
    };
    typedef node* pnode;

    pnode root;

    Segtree(int l, int r) {
      root = new node(l, r);
    }

    void update(pnode& nw, int x, int val) {
      int l = nw->l, r = nw->r, mid = (l + r) / 2;
      if(l == r)
        nw->val = val;
      else {
        assert(l <= x && x <= r);
        pnode& child = x <= mid ? nw->lc : nw->rc;
        if(!child)
          child = new node(x, x, val);
        else if(child->l <= x && x <= child->r)
          update(child, x, val);
        else {
          do {
            if(x <= mid)
              r = mid;
            else
              l = mid + 1;
            mid = (l + r) / 2;
          } while((x <= mid) == (child->l <= mid));
          pnode nxt = new node(l, r);
          if(child->l <= mid)
            nxt->lc = child;
          else
            nxt->rc = child;
          child = nxt;
          update(nxt, x, val);
        }
        nw->val = min(nw->lc ? nw->lc->val : INF,
                      nw->rc ? nw->rc->val : INF);
      }
    }

    int query(pnode& nw, int x1, int x2) {
      if(!nw)
        return INF;
      int& l = nw->l, &r = nw->r;
      if(r < x1 || x2 < l)
        return INF;
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 2 of 25

```
    if(x1 <= l && r <= x2)
      return nw->val;
    int ret = min(query(nw->lc, x1, x2),
                  query(nw->rc, x1, x2));
    return ret;
  }

  void update(int x, int val) {
    assert(root->l <= x && x <= root->r);
    update(root, x, val);
  }

  int query(int l, int r) {
    return query(root, l, r);
  }
};

struct node {
  int l, r;
  Segtree y;
  node* lc, * rc;
  node(int _l, int _r) : l(_l), r(_r), y(0, MAX),
    lc(NULL), rc(NULL) {}
};
typedef node* pnode;

pnode root;

Segtree2D(int l, int r) {
  root = new node(l, r);
}

void update(pnode& nw, int x, int y, int val) {
  int& l = nw->l, &r = nw->r, mid = (l + r) / 2;
  if(l == r)
    nw->y.update(y, val);
  else {
    if(x <= mid) {
      if(!nw->lc)
        nw->lc = new node(l, mid);
      update(nw->lc, x, y, val);
    } else {
      if(!nw->rc)
        nw->rc = new node(mid + 1, r);
      update(nw->rc, x, y, val);
    }
    val = min(nw->lc ? nw->lc->y.query(y, y) : INF,
              nw->rc ? nw->rc->y.query(y, y) : INF);
    nw->y.update(y, val);
  }
}

int query(pnode& nw, int x1, int x2, int y1, int y2) {
  if(!nw)
    return INF;
  int& l = nw->l, &r = nw->r;
  if(r < x1 || x2 < l)
    return INF;
  if(x1 <= l && r <= x2)
    return nw->y.query(y1, y2);
  int ret = min(query(nw->lc, x1, x2, y1, y2),
                query(nw->rc, x1, x2, y1, y2));
  return ret;
}

void update(int x, int y, int val) {
  assert(root->l <= x && x <= root->r);
  update(root, x, y, val);
}

int query(int x1, int x2, int y1, int y2) {
    return query(root, x1, x2, y1, y2);
  }
};
```

2.2 Fenwick RU-RQ

```
void updtRL(int l, int r, ll val) {
  updt(BIT1, l, val), updt(BIT1, r + 1, -val);
  updt(BIT2, l, val * (l - 1)), updt(BIT2, r + 1, -val * r);
}
ll query(int k) {
  return que(BIT1, k) * k - que(BIT2, k);
}
```

2.3 Heavy-Light Decomposition

```
struct HLD {
  int n;
  vector<int> id, size, idx, up, root, st;
  vector<vector<int>> adj, chain;
  SegTree seg;

  HLD(const vector<vector<int>>& edges) :
    n(edges.size()), id(n, -1), size(n, -1), idx(n, -1),
    up(n, -1), adj(edges), seg(n) {
    precompute(0, -1);
    decompose(0, -1);
    int cnt = 0;
    st.resize(chain.size());
    for(int i = 0; i < (int) chain.size(); ++i) {
      st[i] = cnt;
      cnt += chain[i].size();
    }
  }

  void precompute(int pos, int dad) {
    size[pos] = 1;
    up[pos] = dad;
    for(auto& i : adj[pos]) {
      if(i != dad) {
        precompute(i, pos);
        size[pos] += size[i];
      }
    }
  }

  void decompose(int pos, int dad) {
    if(id[pos] == -1) {
      id[pos] = chain.size();
      root.push_back(pos);
      chain.emplace_back();
    }
    idx[pos] = chain[id[pos]].size();
    chain[id[pos]].push_back(pos);
    int mx = 0, heavy = -1;
    for(auto& i : adj[pos]) {
      if(i != dad && size[i] > mx) {
        mx = size[i];
        heavy = i;
      }
    }
    if(heavy != -1)
      id[heavy] = id[pos];
    for(auto& i : adj[pos]) {
      if(i != dad)
        decompose(i, pos);
    }
  }
```

Proof by... forgor
Bina Nusantara University

Page 3 of 25

```cpp
  void update(int ch, int l, int r, int val) {
    seg.update(st[ch] + l, st[ch] + r, val);
  }

  int query(int ch, int l, int r, int val) {
    return seg.query(st[ch] + l, st[ch] + r, val);
  }
};

// how to move from u to v
while(1) {
  if(hld.id[u] == hld.id[v]) {
    if(hld.idx[u] > hld.idx[v])
      swap(u, v);
    hld.update(hld.id[u], hld.idx[u], hld.idx[v], w);
    // or hld.query(hld.id[u], hld.idx[u], hld.idx[v]);
    break;
  }
  if(hld.id[u] < hld.id[v])
    swap(u, v);
  hld.update(hld.id[u], 0, hld.idx[u], w);
  // or hld.query(hld.id[u], 0, hld.idx[u]);
  u = hld.up[hld.root[hld.id[u]]];
}
```

2.4 Li-Chao Tree

```cpp
// max li-chao tree
// works for the range [0, MAX - 1]
// if min li-chao tree:
// replace every call to max() with min() and every > with <
// also replace -INF with INF

struct Func {
  ll m, c;
  ll operator()(ll x) {
    return x * m + c;
  }
};

const int MAX = 1e9 + 1;
const ll INF = 1e18;
const Func NIL = {0, -INF};

struct Node {
  Func f;
  Node* lc;
  Node* rc;

  Node() : f(NIL), lc(nullptr), rc(nullptr) {}
  Node(const Node& n) : f(n.f), lc(nullptr), rc(nullptr) {}
};

Node* root = new Node;

void insert(Func f, Node* cur = root, int l = 0, int r = MAX - 1) {
  int m = l + (r - l) / 2;
  bool left = f(l) > cur->f(l);
  bool mid = f(m) > cur->f(m);
  if(mid)
    swap(f, cur->f);
  if(l != r) {
    if(left != mid) {
      if(!cur->lc)
        cur->lc = new Node(*cur);
      insert(f, cur->lc, l, m);
    } else {
      if(!cur->rc)
        cur->rc = new Node(*cur);
      insert(f, cur->rc, m + 1, r);
    }
  }
}

ll query(ll x, Node* cur = root, int l = 0, int r = MAX - 1) {
  if(!cur)
    return -INF;
  if(l == r)
    return cur->f(x);
  int m = l + (r - l) / 2;
  if(x <= m)
    return max(cur->f(x), query(x, cur->lc, l, m));
  else
    return max(cur->f(x), query(x, cur->rc, m + 1, r));
}
```

2.5 STL PBDS

```cpp
// ost = ordered set
// omp = ordered map
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<class T>
using ost = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
template<class T, class U>
using omp = tree<T, U, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>;
```

2.6 Treap

```cpp
// Complexity: O(log N) for split and merge
//
// empty treap: Treap* tr = nullptr;
// insert v at x: [l, r] = split(tr, x), m = Treap(v), merge lmr
// delete at x: [l, r] = split(tr, x), [m, r] = split(r, 1), merge lr
// lazy prop: propagate every time a node is accessed

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

using Key = int;

struct Treap {
  Key val;
  Treap* left;
  Treap* right;
  int prio, sz;
  Treap() {}
  Treap(int _val);
};

int size(Treap* tr) {
  return tr ? tr->sz : 0;
}

void update(Treap* tr) {
  tr->sz = 1 + size(tr->left) + size(tr->right);
}

Treap::Treap(Key _val) :
  val(_val), left(nullptr), right(nullptr), prio(rng()) {
  update(this);
}

pair<Treap*, Treap*> split(Treap* tr, int sz) {
  if(!tr) return {nullptr, nullptr};
```

```
  int left_sz = size(tr->left);
  if(sz <= left_sz) {
    auto [left, mid] = split(tr->left, sz);
    tr->left = mid;
    update(tr);
    return {left, tr};
  } else {
    auto [mid, right] = split(tr->right, sz - left_sz - 1);
    tr->right = mid;
    update(tr);
    return {tr, right};
  }
}

Treap* merge(Treap* l, Treap* r) {
  if(!l)
    return r;
  if(!r)
    return l;
  if(l->prio < r->prio) {
    l->right = merge(l->right, r);
    update(l);
    return l;
  } else {
    r->left = merge(l, r->left);
    update(r);
    return r;
  }
}
```

2.7 Unordered Map Custom Hash

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
      chrono::steady_clock::now().time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

unordered_map<int, int, custom_hash> umap;
```

2.8 Mo's on Tree

$ST(u) \leq ST(v)$
$P = LCA(u, v)$
If $P = u$, query $[ST(u), ST(v)]$
Else query $[EN(u), ST(v)] + [ST(P), ST(P)]$

2.9 Link-Cut Tree

```
// Represents a forest of unrooted trees. You can add and remove edges
// (as long as the result is still a forest), and check whether two
// nodes are in the same tree.
// Complexity: log(n)

struct Node { // Splay tree. Root's pp contains tree's parent.
  Node* p = 0, * pp = 0, * c[2];
  int sz = 0;
  //
  Node() {
    c[0] = c[1] = 0;
    fix();
  }
  void fix() {
    sz = 1;
    if(c[0]) c[0]->p = this, sz += c[0]->sz;
    if(c[1]) c[1]->p = this, sz += c[1]->sz;
    // (+ update sum of subtree elements etc. if wanted)
  }
  int up() {
    return p ? p->c[1] == this : -1;
  }
  void rot(int i, int b) {
    int h = i ^ b;
    Node* x = c[i], * y = (b == 2 ? x : x->c[h]), * z = (b ? y : x);
    if(y->p = p) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if(b < 2) x->c[h] = y->c[h ^ 1], z->c[h ^ 1] = b ? x : this;
    y->c[i ^ 1] = b ? this : x;
    fix();
    x->fix();
    y->fix();
    if(p) p->fix();
    swap(pp, y->pp);
  }
  // Splay this up to the root. Always finishes without flip set.
  void splay() {
    while(p) {
      int c1 = up(), c2 = p->up();
      if(c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
};
struct LinkCut {
  vector<Node> node;

  LinkCut(int N) : node(N + 1) {}
  void link(int u, int v) { // add an edge u --> v
    assert(!connected(u, v));
    access(&node[u]);
    access(&node[v]);
    node[u].c[0] = &node[v];
    node[v].p = &node[u];
    node[u].fix();
  }
  void cut(int u, int v) { // remove an edge u --> v
    assert(connected(u, v));
    Node* x = &node[v], * top = &node[u];
    access(top);
    top->c[0] = top->c[0]->p = 0;
    top->fix();
  }
  bool connected(int u, int v) { // are u, v in the same tree?
    return root(u) == root(v);
  }
  int root(int u) { // find the root id of node u
    Node* x = &node[u];
    access(x);
    for(; x->c[0]; x = x->c[0]);
    x->splay();
    return (int)((vector<Node>::iterator)x - node.begin());
  }
  // Move u to root aux tree. Return the root of the root aux tree.
  Node* access(Node* u) {
    u->splay();
    Node* last = u;
    if(Node*& x = u->c[1]) {
      x->pp = u;
      x->p = 0;
      x = 0;
      u->fix();
```


Proof by... forgor
Bina Nusantara University

Page 5 of 25

```
    }
    for(Node * pp; (pp = u->pp) && (last = pp);) {
      pp->splay();
      if(pp->c[1]) pp->c[1]->p = 0, pp->c[1]->pp = pp;
      pp->c[1] = u;
      u->p = pp;
      u->pp = 0;
      pp->fix();
      u->splay();
    }
    return last;
  }
  int depth(int u) {
    access(&node[u]);
    return node[u].sz - 1;
  }
  Node* lca(int u, int v) {
    access(&node[u]);
    return access(&node[v]);
  }
};
```

3 Dynamic Programming

3.1 DP Convex Hull

```
/* dp[i] = min k<i {dp[k] + x[i]*m[k]}
   Make sure gradient (m[i]) is either non-increasing if min,
   or non-decreasing if max. x[i] must be non-decreasing. just sort */
int y[N], m[N];
// while this is true, pop back from dq. a=new line, b=last, c=2nd last
bool cekx(int a, int b, int c) {
  // if not enough, change to cross mul
  // if cross mul, beware of negative denominator, and overflow
  return (double)(y[b] - y[a]) / (m[a] - m[b]) <= (double)(y[c] - y[b]) /
         (m[b] - m[c]);
}
```

3.2 DP DNC

```
void f(int rem, int l, int r, int optl, int optr) {
  if(l > r)
    return;
  int mid = l + r >> 1;
  int opt = MOD, optid = mid;
  for(int i = optl; i <= mid && i <= optr; ++i) {
    if(dp[rem - 1][i] + c[i][mid] < opt) {
      opt = dp[rem - 1][i] + c[i][mid];
      optid = i;
    }
  }
  dp[rem][mid] = opt;
  f(rem, l, mid - 1, optl, optid);
  f(rem, mid + 1, r, optid, optr);
  return;
}
rep(i, 1, n)dp[1][i] = c[0][i];
rep(i, 2, k)f(i, i, n, i, n);
```

3.3 DP Knuth-Yao

```
// opt[i+1][j] <= opt[i][j] <= opt[i][j+1]
// dp[i][j] = min{k} dp[i][k]+dp[k][j]+cost[i][j]
for(int k = 0; k <= n; k++) {
  for(int i = 0; i + k <= n; i++) {
    if(k < 2)
      dp[i][i + k] = 0, opt[i][i + k] = i;
    else {
      int sta = opt[i][i + k - 1];
      int end = opt[i + 1][i + k];
      for(int j = sta; j <= end; j++) {
        if(dp[i][j] + dp[j][i + k] + cost[i][i + k] < dp[i][i + k]) {
          dp[i][i + k] = dp[i][j] + dp[j][i + k] + cost[i][i + k];
          opt[i][i + k] = j;
        }
      }
    }
  }
}
```

4 Geometry

4.1 Geometry Template

```
/*
TABLE OF CONTENT
0. Basic Rule
    0.1. Everything is in double
    0.2. Every comparison use EPS
    0.3. Every degree in rad
1. General Double Operation
    1.1. const double EPS=1E-9
    1.2. const double PI=acos(-1.0)
    1.3. const double INFD=1E9
    1.3. between_d(double x,double l,double r)
        check whether x is between l and r inclusive with EPS
    1.4. same_d(double x,double y)
        check whether x=y with EPS
    1.5. dabs(double x)
        absolute value of x
2. Point
    2.1. struct point
        2.1.1. double x,y
            cartesian coordinate of the point
        2.1.2. point()
            default constructor
        2.1.3. point(double _x,double _y)
            constructor, set the point to (_x,_y)
        2.1.4. bool operator< (point other)
            regular pair<double,double> operator < with EPS
        2.1.5. bool operator== (point other)
            regular pair<double,double> operator == with EPS
    2.2. hypot(point P)
        length of hypotenuse of point P to (0,0)
    2.3. e_dist(point P1,point P2)
        euclidean distance from P1 to P2
    2.4. m_dist(point P1,point P2)
        manhattan distance from P1 to P2
    2.5. point rotate(point P,point O,double angle)
        rotate point P from the origin O by angle ccw
3. Vector
    3.1. struct vec
        3.1.1. double x,y
            x and y magnitude of the vector
        3.1.2. vec()
            default constructor
        3.1.3. vec(double _x,double _y)
            constructor, set the vector to (_x,_y)
        3.1.4. vec(point A,point B)
            constructor, set the vector to vector AB (A->B)
*/
/*General Double Operation*/

const double PI = acos(-1.0);
const double INFD = 1E9;
double between_d(double x, double l, double r) {
```

```
  return (min(l, r) <= x + EPS && x <= max(l, r) + EPS);
}
double same_d(double x, double y) {
  return between_d(x, y, y);
}
double dabs(double x) {
  if(x < EPS)
    return -x;
  return x;
}
/*Point*/
struct point {
  double x, y;
  point() {
    x = y = 0.0;
  }
  point(double _x, double _y) {
    x = _x;
    y = _y;
  }
  bool operator< (point other) {
    if(x < other.x + EPS)
      return true;
    if(x + EPS > other.x)
      return false;
    return y < other.y + EPS;
  }
  bool operator== (point other) {
    return same_d(x, other.x) && same_d(y, other.y);
  }
};
double e_dist(point P1, point P2) {
  return hypot(P1.x - P2.x, P1.y - P2.y);
}
double m_dist(point P1, point P2) {
  return dabs(P1.x - P2.x) + dabs(P1.y - P2.y);
}
double pointBetween(point P, point L, point R) {
  return (e_dist(L, P) + e_dist(P, R) == e_dist(L, R));
}
bool collinear(point P, point L,
               point R) { //newly added(luis), cek 3 poin segaris
  return P.x * (L.y - R.y) + L.x * (R.y - P.y) + R.x * (P.y - L.y) ==
         0; // bole gnti “dabs(x)<”EPS
}

/*Vector*/
struct vec {
  double x, y;
  vec() {
    x = y = 0.0;
  }
  vec(double _x, double _y) {
    x = _x;
    y = _y;
  }
  vec(point A) {
    x = A.x;
    y = A.y;
  }
  vec(point A, point B) {
    x = B.x - A.x;
    y = B.y - A.y;
  }
};
vec scale(vec v, double s) {
  return vec(v.x * s, v.y * s);
}
vec flip(vec v) {
  return vec(-v.x, -v.y);
}
double dot(vec u, vec v) {
  return (u.x * v.x + u.y * v.y);
}
double cross(vec u, vec v) {
  return (u.x * v.y - u.y * v.x);
}
double norm_sq(vec v) {
  return (v.x * v.x + v.y * v.y);
}
point translate(point P, vec v) {
  return point(P.x + v.x, P.y + v.y);
}
point rotate(point P, point O, double angle) {
  vec v(O);
  P = translate(P, flip(v));
  return translate(point(P.x * cos(angle) - P.y * sin(angle),
                         P.x * sin(angle) + P.y * cos(angle)), v);
}
point mid(point P, point Q) {
  return point((P.x + Q.x) / 2, (P.y + Q.y) / 2);
}
double angle(point A, point O, point B) {
  vec OA(O, A), OB(O, B);
  return acos(dot(OA, OB) / sqrt(norm_sq(OA) * norm_sq(OB)));
}
int orientation(point P, point Q, point R) {
  vec PQ(P, Q), PR(P, R);
  double c = cross(PQ, PR);
  if(c < -EPS)
    return -1;
  if(c > EPS)
    return 1;
  return 0;
}
/*Line*/
struct line {
  double a, b, c;
  line() {
    a = b = c = 0.0;
  }
  line(double _a, double _b, double _c) {
    a = _a;
    b = _b;
    c = _c;
  }
  line(point P1, point P2) {
    if(P1 < P2)
      swap(P1, P2);
    if(same_d(P1.x, P2.x))
      a = 1.0, b = 0.0, c = -P1.x;
    else
      a = -(P1.y - P2.y) / (P1.x - P2.x), b = 1.0, c = -(a * P1.x) - P1.y;
  }
  line(point P, double slope) {
    if(same_d(slope, INFD))
      a = 1.0, b = 0.0, c = -P.x;
    else
      a = -slope, b = 1.0, c = -(a * P.x) - P.y;
  }
  bool operator== (line other) {
    return same_d(a, other.a) && same_d(b, other.b) && same_d(c, other.c);
  }
  double slope() {
    if(same_d(b, 0.0))
      return INFD;
    return -(a / b);
  }
};
bool paralel(line L1, line L2) {
  return same_d(L1.a, L2.a) && same_d(L1.b, L2.b);
}
```


Proof by... forgor
Bina Nusantara University

Page 7 of 25

Owen Djonatan, Luis Anthonie Alkins, Richard Alison


```
bool intersection(line L1, line L2, point& P) {
  if(paralel(L1, L2))
    return false;
  P.x = (L2.b * L1.c - L1.b * L2.c) / (L2.a * L1.b - L1.a * L2.b);
  if(same_d(L1.b, 0.0))
    P.y = -(L2.a * P.x + L2.c);
  else
    P.y = -(L1.a * P.x + L1.c);
  return true;
}
double pointToLine(point P, point A, point B, point& C) {
  vec AP(A, P), AB(A, B);
  double u = dot(AP, AB) / norm_sq(AB);
  C = translate(A, scale(AB, u));
  return e_dist(P, C);
}
double lineToLine(line L1, line L2) {
  if(!paralel(L1, L2))
    return 0.0;
  return dabs(L2.c - L1.c) / sqrt(L1.a * L1.a + L1.b * L1.b);
}
/*Line Segment*/
struct segment {
  point P, Q;
  line L;
  segment() {
    point T1;
    P = Q = T1;
    line T2;
    L = T2;
  }
  segment(point _P, point _Q) {
    P = _P;
    Q = _Q;
    if(Q < P)
      swap(P, Q);
    line T(P, Q);
    L = T;
  }
  bool operator== (segment other) {
    return P == other.P && Q == other.Q;
  }
};
bool onSegment(point P, segment S) {
  if(orientation(S.P, S.Q, P) != 0)
    return false;
  return between_d(P.x, S.P.x, S.Q.x) && between_d(P.y, S.P.y, S.Q.y);
}
bool s_intersection(segment S1, segment S2) {
  double o1 = orientation(S1.P, S1.Q, S2.P);
  double o2 = orientation(S1.P, S1.Q, S2.Q);
  double o3 = orientation(S2.P, S2.Q, S1.P);
  double o4 = orientation(S2.P, S2.Q, S1.Q);
  if(o1 != o2 && o3 != o4)
    return true;
  if(o1 == 0 && onSegment(S2.P, S1))
    return true;
  if(o2 == 0 && onSegment(S2.Q, S1))
    return true;
  if(o3 == 0 && onSegment(S1.P, S2))
    return true;
  if(o4 == 0 && onSegment(S1.Q, S2))
    return true;
  return false;
}
double pointToSegment(point P, point A, point B, point& C) {
  vec AP(A, P), AB(A, B);
  double u = dot(AP, AB) / norm_sq(AB);
  if(u < EPS) {
    C = A;
    return e_dist(P, A);
  }
  if(u + EPS > 1.0) {
    C = B;
    return e_dist(P, B);
  }
  return pointToLine(P, A, B, C);
}
double segmentToSegment(segment S1, segment S2) {
  if(s_intersection(S1, S2))
    return 0.0;
  double ret = INFD;
  point dummy;
  ret = min(ret, pointToSegment(S1.P, S2.P, S2.Q, dummy));
  ret = min(ret, pointToSegment(S1.Q, S2.P, S2.Q, dummy));
  ret = min(ret, pointToSegment(S2.P, S1.P, S1.Q, dummy));
  ret = min(ret, pointToSegment(S2.Q, S1.P, S1.Q, dummy));
  return ret;
}
/*Circle*/
struct circle {
  point P;
  double r;
  circle() {
    point P1;
    P = P1;
    r = 0.0;
  }
  circle(point _P, double _r) {
    P = _P;
    r = _r;
  }
  circle(point P1, point P2) {
    P = mid(P1, P2);
    r = e_dist(P, P1);
  }
  circle(point P1, point P2, point P3) {
    vector<point> T;
    T.clear();
    T.pb(P1);
    T.pb(P2);
    T.pb(P3);
    sort(T.begin(), T.end());
    P1 = T[0];
    P2 = T[1];
    P3 = T[2];
    point M1, M2;
    M1 = mid(P1, P2);
    M2 = mid(P2, P3);
    point Q2, Q3;
    Q2 = rotate(P2, P1, PI / 2);
    Q3 = rotate(P3, P2, PI / 2);
    vec P1Q2(P1, Q2), P2Q3(P2, Q3);
    point M3, M4;
    M3 = translate(M1, P1Q2);
    M4 = translate(M2, P2Q3);
    line L1(M1, M3), L2(M2, M4);
    intersection(L1, L2, P);
    r = e_dist(P, P1);
  }
  bool operator==(circle other) {
    return (P == other.P && same_d(r, other.r));
  }
};
bool insideCircle(point P, circle C) {
  return e_dist(P, C.P) <= C.r + EPS;
}
bool c_intersection(circle C1, circle C2, point& P1, point& P2) {
  double d = e_dist(C1.P, C2.P);
  if(d > C1.r + C2.r) {
    return false;  //d+EPS kalo butuh
  }
```

Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 8 of 25

```
  if(d < dabs(C1.r - C2.r) + EPS)
    return false;
  double x1 = C1.P.x, y1 = C1.P.y, r1 = C1.r, x2 = C2.P.x, y2 = C2.P.y, r2 = C2.r;
  double a = (r1 * r1 - r2 * r2 + d * d) / (2 * d), h = sqrt(r1 * r1 - a * a);
  point T(x1 + a * (x2 - x1) / d, y1 + a * (y2 - y1) / d);
  P1 = point(T.x - h * (y2 - y1) / d, T.y + h * (x2 - x1) / d);
  P2 = point(T.x + h * (y2 - y1) / d, T.y - h * (x2 - x1) / d);
  return true;
}
bool lc_intersection(line L, circle O, point& P1, point& P2) {
  double a = L.a, b = L.b, c = L.c, x = O.P.x, y = O.P.y, r = O.r;
  double A = a * a + b * b, B = 2 * a * b * y - 2 * a * c - 2 * b * b * x,
         C = b * b * x * x + b * b * y * y - 2 * b * c * y + c * c - b * b * r * r;
  double D = B * B - 4 * A * C;
  point T1, T2;
  if(same_d(b, 0.0)) {
    T1.x = c / a;
    if(dabs(x - T1.x) + EPS > r)
      return false;
    if(same_d(T1.x - r - x, 0.0) || same_d(T1.x + r - x, 0.0)) {
      P1 = P2 = point(T1.x, y);
      return true;
    }
    double dx = dabs(T1.x - x), dy = sqrt(r * r - dx * dx);
    P1 = point(T1.x, y - dy);
    P2 = point(T1.x, y + dy);
    return true;
  }
  if(same_d(D, 0.0)) {
    T1.x = -B / (2 * A);
    T1.y = (c - a * T1.x) / b;
    P1 = P2 = T1;
    return true;
  }
  if(D < EPS)
    return false;
  D = sqrt(D);
  T1.x = (-B - D) / (2 * A);
  T1.y = (c - a * T1.x) / b;
  P1 = T1;
  T2.x = (-B + D) / (2 * A);
  T2.y = (c - a * T2.x) / b;
  P2 = T2;
  return true;
}
bool sc_intersection(segment S, circle C, point& P1, point& P2) {
  bool cek = lc_intersection(S.L, C, P1, P2);
  if(!cek)
    return false;
  double x1 = S.P.x, y1 = S.P.y, x2 = S.Q.x, y2 = S.Q.y;
  bool b1 = between_d(P1.x, x1, x2) && between_d(P1.y, y1, y2);
  bool b2 = between_d(P2.x, x1, x2) && between_d(P2.y, y1, y2);
  if(P1 == P2)
    return b1;
  if(b1 || b2) {
    if(!b1)
      P1 = P2;
    if(!b2)
      P2 = P1;
    return true;
  }
  return false;
}
/*Triangle*/
double t_perimeter(point A, point B, point C) {
  return e_dist(A, B) + e_dist(B, C) + e_dist(C, A);
}
double t_area(point A, point B, point C) {
  double s = t_perimeter(A, B, C) / 2;
  double ab = e_dist(A, B), bc = e_dist(B, C), ac = e_dist(C, A);
  return sqrt(s * (s - ab) * (s - bc) * (s - ac));
}
circle t_inCircle(point A, point B, point C) {
  vector<point> T;
  T.clear();
  T.pb(A);
  T.pb(B);
  T.pb(C);
  sort(T.begin(), T.end());
  A = T[0];
  B = T[1];
  C = T[2];
  double r = t_area(A, B, C) / (t_perimeter(A, B, C) / 2);
  double ratio = e_dist(A, B) / e_dist(A, C);
  vec BC(B, C);
  BC = scale(BC, ratio / (1 + ratio));
  point P;
  P = translate(B, BC);
  line AP1(A, P);
  ratio = e_dist(B, A) / e_dist(B, C);
  vec AC(A, C);
  AC = scale(AC, ratio / (1 + ratio));
  P = translate(A, AC);
  line BP2(B, P);
  intersection(AP1, BP2, P);
  return circle(P, r);
}
circle t_outCircle(point A, point B, point C) {
  return circle(A, B, C);
}
/*Polygon*/
struct polygon {
  vector<point> P;
  polygon() {
    P.clear();
  }
  polygon(vector<point>& _P) {
    P = _P;
  }
};
bool rayCast(point P, polygon& A) {
  point Q(P.x, 10000);
  line cast(P, Q);
  int cnt = 0;
  FOR(i, (int)(A.P.size()) - 1) {
    line temp(A.P[i], A.P[i + 1]);
    point I;
    bool B = intersection(cast, temp, I);
    if(!B)
      continue;
    else if(I == A.P[i] || I == A.P[i + 1])
      continue;
    else if(pointBetween(I, A.P[i], A.P[i + 1]) && pointBetween(I, P, Q))
      cnt++;
  }
  return cnt % 2 == 1;
}
// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u + v), (p.y * v + q.y * u) / (u + v));
}
// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, const vector<point>& Q) {
  vector<point> P;
  for(int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b), toVec(a, Q[i]));
```


Proof by... forgor
Bina Nusantara University

Page 9 of 25

```
    double left2 = 0;
    if(i != (int)Q.size() - 1)
      left2 = cross(toVec(a, b), toVec(a, Q[i + 1]));
    if(left1 > -EPS)
      P.push_back(Q[i]);
    if(left1 * left2 < -EPS)
      P.push_back(lineIntersectSeg(Q[i], Q[i + 1], a, b));
  }
  if(!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front());
  return P;
}
circle minCoverCircle(polygon& A) {
  vector<point> p = A.P;
  point c;
  circle ret;
  double cr = 0.0;
  int i, j, k;
  c = p[0];
  for(i = 1; i < p.size(); i++) {
    if(e_dist(p[i], c) >= cr + EPS) {
      c = p[i], cr = 0;
      ret = circle(c, cr);
      for(j = 0; j < i; j++) {
        if(e_dist(p[j], c) >= cr + EPS) {
          c = mid(p[i], p[j]);
          cr = e_dist(p[i], c);
          ret = circle(c, cr);
          for(k = 0; k < j; k++) {
            if(e_dist(p[k], c) >= cr + EPS) {
              ret = circle(p[i], p[j], p[k]);
              c = ret.P;
              cr = ret.r;
            }
          }
        }
      }
    }
  }
  return ret;
}
/*Geometry Algorithm*/
double DP[110][110];
double minCostPolygonTriangulation(polygon& A) {
  if(A.P.size() < 3)
    return 0;
  FOR(i, A.P.size()) {
    for(int j = 0, k = i; k < A.P.size(); j++, k++) {
      if(k < j + 2)
        DP[j][k] = 0.0;
      else {
        DP[j][k] = INFD;
        REP(l, j + 1, k - 1) {
          double cost = e_dist(A.P[j], A.P[k]) + e_dist(A.P[k], A.P[l]) + e_dist(A.P[l↩
              ],
                      A.P[j]);
          DP[j][k] = min(DP[j][k], DP[j][l] + DP[l][k] + cost);
        }
      }
    }
  }
  return DP[0][A.P.size() - 1];
}
```

4.2 Convex Hull

```
typedef double TD;                 // for precision shits
namespace GEOM {
  typedef pair<TD, TD> Pt;         // vector and points
  const TD EPS = 1e-9;
  const TD maxD = 1e9;
  TD cross(Pt a, Pt b, Pt c) {     // right hand rule
    TD v1 = a.first - c.first;     // (a-c) X (b-c)
    TD v2 = a.second - c.second;
    TD u1 = b.first - c.first;
    TD u2 = b.second - c.second;
    return v1 * u2 - v2 * u1;
  }
  TD cross(Pt a, Pt b) {           // a X b
    return a.first * b.second - a.second * b.first;
  }
  TD dot(Pt a, Pt b, Pt c) {       // (a-c) . (b-c)
    TD v1 = a.first - c.first;
    TD v2 = a.second - c.second;
    TD u1 = b.first - c.first;
    TD u2 = b.second - c.second;
    return v1 * u1 + v2 * u2;
  }
  TD dot(Pt a, Pt b) {             // a . b
    return a.first * b.first + a.second * b.second;
  }
  TD dist(Pt a, Pt b) {
    return sqrt((a.first - b.first) * (a.first - b.first) +
                (a.second - b.second) * (a.second - b.second));
  }
  TD shoelaceX2(vector<Pt>& convHull) {
    TD ret = 0;
    for(int i = 0, n = convHull.size(); i < n; i++)
      ret += cross(convHull[i], convHull[(i + 1) % n]);
    return ret;
  }
  vector<Pt> createConvexHull(vector<Pt>& points) {
    sort(points.begin(), points.end());
    vector<Pt> ret;
    for(int i = 0; i < points.size(); i++) {
      while(ret.size() > 1 &&
            cross(points[i], ret[ret.size() - 1], ret[ret.size() - 2]) < -EPS)
        ret.pop_back();
      ret.push_back(points[i]);
    }
    for(int i = points.size() - 2, sz = ret.size(); i >= 0; i--) {
      while(ret.size() > sz &&
            cross(points[i], ret[ret.size() - 1], ret[ret.size() - 2]) < -EPS)
        ret.pop_back();
      if(i == 0)
        break;
      ret.push_back(points[i]);
    }
    return ret;
  bool isInside(Pt pv, vector<Pt>& x) { //using winding number
    int n = x.size(), wn = 0;
    x.push_back(x[0]);
    for(int i = 0; i < n; ++i) {
      if(((x[i + 1].first <= pv.first && x[i].first >= pv.first) ||
          (x[i + 1].first >= pv.first && x[i].first <= pv.first)) &&
          ((x[i + 1].second <= pv.second && x[i].second >= pv.second) ||
           (x[i + 1].second >= pv.second && x[i].second <= pv.second))) {
        if(cross(x[i], x[i + 1], pv) == 0) {
          x.pop_back();
          return true;
        }
      }
    }
    for(int i = 0; i < n; ++i) {
      if(x[i].second <= pv.second) {
        if(x[i + 1].second > pv.second && cross(x[i], x[i + 1], pv) > 0)
          ++wn;
      } else if(x[i + 1].second <= pv.second && cross(x[i], x[i + 1], pv) < 0)
        --wn;
    }
    x.pop_back();
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 10 of 25

```
      return wn != 0;
    }
  }
  bool isInside(Pt pv, vector<Pt>& x) { //using winding number
    int n = x.size(), wn = 0;
    x.push_back(x[0]);
    for(int i = 0; i < n; ++i) {
      if(((x[i + 1].first <= pv.first && x[i].first >= pv.first) ||
          (x[i + 1].first >= pv.first && x[i].first <= pv.first)) &&
          ((x[i + 1].second <= pv.second && x[i].second >= pv.second) ||
           (x[i + 1].second >= pv.second && x[i].second <= pv.second))) {
        if(cross(x[i], x[i + 1], pv) == 0) {
          x.pop_back();
          return true;
        }
      }
    }
    for(int i = 0; i < n; ++i) {
      if(x[i].second <= pv.second) {
        if(x[i + 1].second > pv.second && cross(x[i], x[i + 1], pv) > 0)
          ++wn;
      } else if(x[i + 1].second <= pv.second && cross(x[i], x[i + 1], pv) < 0)
        --wn;
    }
    x.pop_back();
    return wn != 0;
  }
}
```

4.3 Closest Pair of Points

```
#define fi first
#define se second
typedef pair<int, int> pii;
struct Point {
  int x, y, id;
};
int compareX(const void* a, const void* b) {
  Point* p1 = (Point*)a, * p2 = (Point*)b;
  return (p1->x - p2->x);
}
int compareY(const void* a, const void* b) {
  Point* p1 = (Point*)a, * p2 = (Point*)b;
  return (p1->y - p2->y);
}
double dist(Point p1, Point p2) {
  return sqrt((double)(p1.x - p2.x) * (p1.x - p2.x) +
              (double)(p1.y - p2.y) * (p1.y - p2.y)
             );
}
pair<pii, double> bruteForce(Point P[], int n) {
  double min = 1e8;
  pii ret = pii(-1, -1);
  for(int i = 0; i < n; ++i)
    for(int j = i + 1; j < n; ++j)
      if(dist(P[i], P[j]) < min) {
        ret = pii(P[i].id, P[j].id);
        min = dist(P[i], P[j]);
      }
  return pair<pii, double> (ret, min);
}
pair<pii, double> getmin(pair<pii, double> x, pair<pii, double> y) {
  if(x.fi.fi == -1 && x.fi.se == -1)
    return y;
  if(y.fi.fi == -1 && y.fi.se == -1)
    return x;
  return (x.se < y.se) ? x : y;
}
pair<pii, double> stripClosest(Point strip[], int size, double d) {
  double min = d;
  pii ret = pii(-1, -1);
  qsort(strip, size, sizeof(Point), compareY);
  for(int i = 0; i < size; ++i)
    for(int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j)
      if(dist(strip[i], strip[j]) < min) {
        ret = pii(strip[i].id, strip[j].id);
        min = dist(strip[i], strip[j]);
      }
  return pair<pii, double>(ret, min);
}
pair<pii, double> closestUtil(Point P[], int n) {
  if(n <= 3)
    return bruteForce(P, n);
  int mid = n / 2;
  Point midPoint = P[mid];
  pair<pii, double> dl = closestUtil(P, mid);
  pair<pii, double> dr = closestUtil(P + mid, n - mid);
  pair<pii, double> d = getmin(dl, dr);
  Point strip[n];
  int j = 0;
  for(int i = 0; i < n; i++)
    if(abs(P[i].x - midPoint.x) < d.second)
      strip[j] = P[i], j++;
  return getmin(d, stripClosest(strip, j, d.second));
}
pair<pii, double> closest(Point P[], int n) {
  qsort(P, n, sizeof(Point), compareX);
  return closestUtil(P, n);
}
Point P[50005];
int main() {
  int n;
  scanf("%d", &n);
  for(int a = 0; a < n; a++) {
    scanf("%d%d", &P[a].x, &P[a].y);
    P[a].id = a;
  }
  pair<pii, double> hasil = closest(P, n);
  if(hasil.fi.fi > hasil.fi.se)
    swap(hasil.fi.fi, hasil.fi.se);
  printf("%d %d %.6lf\n", hasil.fi.fi, hasil.fi.se, hasil.se);
  return 0;
}
```

4.4 Smallest Enclosing Circle

```
// welzl's algo to find the 2d minimum enclosing circle of a set of points
// expected O(N)
// directions: remove duplicates and shuffle points, then call welzl(points)

struct Point {
  double x;
  double y;
};

struct Circle {
  double x, y, r;
  Circle() {}
  Circle(double _x, double _y, double _r): x(_x), y(_y), r(_r) {}
};

Circle trivial(const vector<Point>& r) {
  if(r.size() == 0)
    return Circle(0, 0, -1);
  else if(r.size() == 1)
    return Circle(r[0].x, r[0].y, 0);
  else if(r.size() == 2) {
    double cx = (r[0].x + r[1].x) / 2.0, cy = (r[0].y + r[1].y) / 2.0;
    double rad = hypot(r[0].x - r[1].x, r[0].y - r[1].y) / 2.0;
    return Circle(cx, cy, rad);
```

```
  } else {
    double x0 = r[0].x, x1 = r[1].x, x2 = r[2].x;
    double y0 = r[0].y, y1 = r[1].y, y2 = r[2].y;
    double d = (x0 - x2) * (y1 - y2) - (x1 - x2) * (y0 - y2);
    double cx = (((x0 - x2) * (x0 + x2) + (y0 - y2) * (y0 + y2)) / 2 *
                (y1 - y2) - ((x1 - x2) * (x1 + x2) + (y1 - y2) * (y1 + y2)) / 2
                * (y0 - y2)) / d;
    double cy = (((x1 - x2) * (x1 + x2) + (y1 - y2) * (y1 + y2)) / 2 *
                (x0 - x2) - ((x0 - x2) * (x0 + x2) + (y0 - y2) * (y0 + y2)) / 2
                * (x1 - x2)) / d;
    return Circle(cx, cy, hypot(x0 - cx, y0 - cy));
  }
}

// SHUFFLE THE POINTS FIRST!!!!!!
Circle welzl(const vector<Point>& p, int idx = 0, vector<Point> r = {}) {
  if(idx == (int) p.size() || r.size() == 3)
    return trivial(r);
  Circle d = welzl(p, idx + 1, r);
  if(hypot(p[idx].x - d.x, p[idx].y - d.y) > d.r) {
    r.push_back(p[idx]);
    d = welzl(p, idx + 1, r);
  }
  return d;
}
```

4.5 Sutherland-Hodgman Algorithm

```
// Complexity: linear time
// Ada 2 poligon, cari poligon intersectionnya
// poly_point = hasilnya, clipper = pemotongnya
#include<bits/stdc++.h>
using namespace std;

const double EPS = 1e-9;

struct point {
  double x, y;
  point(double _x, double _y): x(_x), y(_y) {}
};
struct vec {
  double x, y;
  vec(double _x, double _y): x(_x), y(_y) {}
};

point pivot(0, 0);
vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y);
}
double dist(point a, point b) {
  return hypot(a.x - b.x, a.y - b.y);
}
double cross(vec a, vec b) {
  return a.x * b.y - a.y * b.x;
}
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0;
}
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
}
bool lies(point a, point b, point c) {
  if((c.x >= min(a.x, b.x) && c.x <= max(a.x, b.x)) &&
      (c.y >= min(a.y, b.y) && c.y <= max(a.y, b.y)))
    return true;
  else
    return false;
}
bool anglecmp(point a, point b) {
  if(collinear(pivot, a, b))
    return dist(pivot, a) < dist(pivot, b);
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
}

point intersect(point s1, point e1, point s2, point e2) {
  double x1, x2, x3, x4, y1, y2, y3, y4;
  x1 = s1.x;
  y1 = s1.y;
  x2 = e1.x;
  y2 = e1.y;
  x3 = s2.x;
  y3 = s2.y;
  x4 = e2.x;
  y4 = e2.y;
  double num1 = (x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 * x4);
  double num2 = (x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 * x4);
  double den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
  double new_x = num1 / den;
  double new_y = num2 / den;
  return point(new_x, new_y);
}

void clip(vector <point>& poly_points, point point1, point point2) {
  vector <point> new_points;
  new_points.clear();
  for(int i = 0; i < poly_points.size(); i++) {
    int k = (i + 1) % poly_points.size();
    double i_pos = ccw(point1, point2, poly_points[i]);
    double k_pos = ccw(point1, point2, poly_points[k]);
    //in in
    if(i_pos <= 0  && k_pos <= 0)
      new_points.push_back(poly_points[k]);
    //out in
    else if(i_pos > 0  && k_pos <= 0) {
      new_points.push_back(intersect(point1, point2, poly_points[i],
                                     poly_points[k]));
      new_points.push_back(poly_points[k]);
    }
    // in out
    else if(i_pos <= 0  && k_pos > 0) {
      new_points.push_back(intersect(point1, point2, poly_points[i],
                                     poly_points[k]));
    }
    //out out
    else {
    }
  }
  poly_points.clear();
  for(int i = 0; i < new_points.size(); i++)
    poly_points.push_back(new_points[i]);
}
double area(const vector <point>& P) {
  double result = 0.0;
  double x1, y1, x2, y2;
  for(int i = 0; i < P.size() - 1; i++) {
    x1 = P[i].x;
    y1 = P[i].y;
    x2 = P[i + 1].x;
    y2 = P[i + 1].y;
    result += (x1 * y2 - x2 * y1);
  }
  return fabs(result) / 2;
}
void suthHodgClip(vector <point>& poly_points, vector <point> clipper_points) {
  for(int i = 0; i < clipper_points.size(); i++) {
    int k = (i + 1) % clipper_points.size();
    clip(poly_points, clipper_points[i], clipper_points[k]);
  }
}
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 12 of 25

```
vector<point> sortku(vector<point> P) {
  int P0 = 0;
  int i;
  for(i = 1; i < 3; i++) {
    if(P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;
  }
  point temp = P[0];
  P[0] = P[P0];
  P[P0] = temp;
  pivot = P[0];
  sort(++P.begin(), P.end(), anglecmp);
  reverse(++P.begin(), P.end());
  return P;
}
int main {
  clipper_points = sortku(clipper_points);
  suthHodgClip(poly_points, clipper_points);
}
```

4.6 Centroid of Polygon

$C_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i)$
$C_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i\ y_{i+1} - x_{i+1}\ y_i)$

4.7 Pick Theorem

A: Area of a simply closed lattice polygon
B: Number of lattice points on the edges
I: Number of points in the interior
$A = I + \frac{B}{2} - 1$

5 Graphs

5.1 Articulation Point and Bridge

```
// gr -> adj list
// vector vis, low -> initialize to -1
// int timer -> initialize to 0
void dfs(int pos, int dad = -1) {
  vis[pos] = low[pos] = timer++;
  int kids = 0;
  for(auto& i : gr[pos]) {
    if(i == dad)
      continue;
    if(vis[i] >= 0)
      low[pos] = min(low[pos], vis[i]);
    else {
      dfs(i, pos);
      low[pos] = min(low[pos], low[i]);
      if(low[i] > vis[pos])
        is_bridge(pos, i)
        if(low[i] >= vis[pos] && dad >= 0)
          is_articulation_point(pos)
          ++kids;
    }
  }
  if(dad == -1 && kids > 1)
    is_articulation_point(pos)
  }
```

5.2 SCC and Strong Orientation

```
#define N 10020
vector<int> adj[N];
bool vis[N], ins[N];
int disc[N], low[N], gr[N];
stack<int> st;
int id, grid;
void scc(int cur, int par) {
  disc[cur] = low[cur] = ++id;
  vis[cur] = ins[cur] = 1;
  st.push(cur);
  for(int to : adj[cur]) {
    //if (to==par) continue; // ini untuk SO(scc undirected)
    if(!vis[to])
      scc(to, cur);
    if(ins[to])
      low[cur] = min(low[cur], low[to]);
  }
  if(low[cur] == disc[cur]) {
    grid++; // group id
    while(ins[cur]) {
      gr[st.tp] = grid;
      ins[st.tp] = 0;
      st.pop();
    }
  }
}
```

5.3 Dinic's Maximum Flow

```
// O(VE log(max_flow)) if scaling == 1
// O((V + E) sqrt(E)) if unit graph (turn scaling off)
// O((V + E) sqrt(V)) if bipartite matching (turn scaling off)
// indices are 0-based
const ll INF = 1e18;

struct Dinic {
  struct Edge {
    int v;
    ll cap, flow;
    Edge(int _v, ll _cap): v(_v), cap(_cap), flow(0) {}
  };

  int n;
  ll lim;
  vector<vector<int>> gr;
  vector<Edge> e;
  vector<int> idx, lv;

  bool has_path(int s, int t) {
    queue<int> q;
    q.push(s);
    lv.assign(n, -1);
    lv[s] = 0;
    while(!q.empty()) {
      int c = q.front();
      q.pop();
      if(c == t)
        break;
      for(auto& i : gr[c]) {
        ll cur_flow = e[i].cap - e[i].flow;
        if(lv[e[i].v] == -1 && cur_flow >= lim) {
          lv[e[i].v] = lv[c] + 1;
          q.push(e[i].v);
        }
      }
    }
    return lv[t] != -1;
  }

  ll get_flow(int s, int t, ll left) {
    if(!left || s == t)
      return left;
    while(idx[s] < (int) gr[s].size()) {
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 13 of 25

```
      int i = gr[s][idx[s]];
      if(lv[e[i].v] == lv[s] + 1) {
        ll add = get_flow(e[i].v, t, min(left, e[i].cap - e[i].flow));
        if(add) {
          e[i].flow += add;
          e[i ^ 1].flow -= add;
          return add;
        }
      }
      ++idx[s];
    }
    return 0;
  }

  Dinic(int vertices, bool scaling = 1) : // toggle scaling here
    n(vertices), lim(scaling ? 1 << 30 : 1), gr(n) {}

  void add_edge(int from, int to, ll cap, bool directed = 1) {
    gr[from].push_back(e.size());
    e.emplace_back(to, cap);
    gr[to].push_back(e.size());
    e.emplace_back(from, directed ? 0 : cap);
  }

  ll get_max_flow(int s, int t) { // call this
    ll res = 0;
    while(lim) {  // scaling
      while(has_path(s, t)) {
        idx.assign(n, 0);
        while(ll add = get_flow(s, t, INF))
          res += add;
      }
      lim >>= 1;
    }
    return res;
  }
};
```

5.4 Minimum Cost Maximum Flow

```
using FlowT = ll;
using CostT = ll;

const FlowT F_INF = 1e18;
const CostT C_INF = 1e18;
const int MAX_V = 1e5 + 5;
const int MAX_E = 1e6 + 5;

namespace MCMF {
  int n, E;
  int adj[MAX_E], nxt[MAX_E], lst[MAX_V], frm[MAX_V], vis[MAX_V];
  FlowT cap[MAX_E], flw[MAX_E], totalFlow;
  CostT cst[MAX_E], dst[MAX_V], totalCost;

  void init(int _n) {
    n = _n;
    fill_n(lst, n, -1), E = 0;
  }
  void add(int u, int v, FlowT ca, CostT co) {
    adj[E] = v, cap[E] = ca, flw[E] = 0, cst[E] = +co;
    nxt[E] = lst[u], lst[u] = E++;
    adj[E] = u, cap[E] =  0, flw[E] = 0, cst[E] = -co;
    nxt[E] = lst[v], lst[v] = E++;
  }
  int spfa(int s, int t) {
    fill_n(dst, n, C_INF), dst[s] = 0;
    queue<int> que;
    que.push(s);
    while(que.size()) {
      int u = que.front();
      que.pop();
      for(int e = lst[u]; e != -1; e = nxt[e])
        if(flw[e] < cap[e]) {
          int v = adj[e];
          if(dst[v] > dst[u] + cst[e]) {
            dst[v] = dst[u] + cst[e];
            frm[v] = e;
            if(!vis[v]) {
              vis[v] = 1;
              que.push(v);
            }
          }
        }
      vis[u] = 0;
    }
    return dst[t] < C_INF;
  }
  pair<FlowT, CostT> solve(int s, int t) {
    totalCost = 0, totalFlow = 0;
    while(1) {
      if(!spfa(s, t))
        break;
      FlowT mn = F_INF;
      for(int v = t, e = frm[v]; v != s; v = adj[e ^ 1], e = frm[v])
        mn = min(mn, cap[e] - flw[e]);
      for(int v = t, e = frm[v]; v != s; v = adj[e ^ 1], e = frm[v]) {
        flw[e] += mn;
        flw[e ^ 1] -= mn;
      }
      totalFlow += mn;
      totalCost += mn * dst[t];
    }
    return {totalFlow, totalCost};
  }
};
```

5.5 Flows with Demands

```
let S0 be the source and T0 be the original sink
1. add 2 additional nodes, call them S1 and T1
2. connect S0 to nodes normally
3. connect nodes to T0 normally
4. for each edge(U, V), cap = original cap - demand
5. for each node N :
   1. add an edge(S1, N), cap = sum of inward demand to N
   2. add an edge(N, T1), cap = sum of outward demand from N
6. add an edge(T0, S0), cap = INF
7. the above is not a typo!
8. run max flow normally
9. for each edge(S1, V) and (U, T1), check if flow == cap
if step #9 fails, then it is not possible to satisfy the given demand
```

Mathematically, let $d(e)$ be the demand of edge e. Let V be the set of every vertex in the graph.

- $c'(S_1, v) = \sum_{u \in V} d(u, v)$ for each edge (s', v).
- $c'(v, T_1) = \sum_{v \in V} d(v, w)$ for each edge (v, t').
- $c'(u, v) = c(u, v) - d(u, v)$ for each edge (u, v) in the old network.
- $c'(T_0, S_0) = \infty$

5.6 Hungarian

```
template <typename TD> struct Hungarian {
  TD INF = 1e9;  //max_inf
  int n;
  vector<vector<TD> > adj; // cost[left][right]
  vector<TD> hl, hr, slk;
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 14 of 25

```
  vector<int> fl, fr, vl, vr, pre;
  deque<int> q;
  Hungarian(int _n) {
    n = _n;
    adj = vector<vector<TD> >(n, vector<TD> (n, 0));
  }
  int check(int i) {
    if(vl[i] = 1, fl[i] != -1)
      return q.push_back(fl[i]), vr[fl[i]] = 1;
    while(i != -1)
      swap(i, fr[fl[i] = pre[i]]);
    return 0;
  }
  void bfs(int s) {
    slk.assign(n, INF);
    vl.assign(n, 0);
    vr = vl;
    q.assign(vr[s] = 1, s);
    for(TD d;;) {
      for(; !q.empty(); q.pop_front()) {
        for(int i = 0, j = q.front(); i < n; i++) {
          if(d = hl[i] + hr[j] - adj[i][j], !vl[i] && d <= slk[i]) {
            if(pre[i] = j, d)
              slk[i] = d;
            else if(!check(i))
              return;
          }
        }
      }
      d = INF;
      for(int i = 0; i < n; i++) if(!vl[i] && d > slk[i])
          d = slk[i];
      for(int i = 0; i < n; i++) {
        if(vl[i])
          hl[i] += d;
        else
          slk[i] -= d;
        if(vr[i])
          hr[i] -= d;
      }
      for(int i = 0; i < n; i++) if(!vl[i] && !slk[i] && !check(i))
          return;
    }
  }
  TD solve() {
    fl.assign(n, -1);
    fr = fl;
    hl.assign(n, 0);
    hr = hl;
    pre.assign(n, 0);
    for(int i = 0; i < n; i++)
      hl[i] = *max_element(adj[i].begin(), adj[i].begin() + n);
    for(int i = 0; i < n; i++)
      bfs(i);
    TD ret = 0;
    for(int i = 0; i < n; i++) if(adj[i][fl[i]])
        ret += adj[i][fl[i]];
    return ret;
  }
}; //i wiLL be matched with fl[i]
```

5.7 Edmonds' Blossom

```
// Maximum matching on general graphs in O(V^2 E)
// Indices are 1-based
// Stolen from ko_osaga's cheatsheet
struct Blossom {
  vector<int> vis, dad, orig, match, aux;
  vector<vector<int>> conn;
  int t, N;
  queue<int> Q;

  void augment(int u, int v) {
    int pv = v;
    do {
      pv = dad[v];
      int nv = match[pv];
      match[v] = pv;
      match[pv] = v;
      v = nv;
    } while(u != pv);
  }

  int lca(int v, int w) {
    ++t;
    while(true) {
      if(v) {
        if(aux[v] == t)
          return v;
        aux[v] = t;
        v = orig[dad[match[v]]];
      }
      swap(v, w);
    }
  }

  void blossom(int v, int w, int a) {
    while(orig[v] != a) {
      dad[v] = w;
      w = match[v];
      if(vis[w] == 1) {
        Q.push(w);
        vis[w] = 0;
      }
      orig[v] = orig[w] = a;
      v = dad[w];
    }
  }

  bool bfs(int u) {
    fill(vis.begin(), vis.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    Q = queue<int>();
    Q.push(u);
    vis[u] = 0;
    while(!Q.empty()) {
      int v = Q.front();
      Q.pop();
      for(int x : conn[v]) {
        if(vis[x] == -1) {
          dad[x] = v;
          vis[x] = 1;
          if(!match[x]) {
            augment(u, x);
            return 1;
          }
          Q.push(match[x]);
          vis[match[x]] = 0;
        } else if(vis[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a);
          blossom(v, x, a);
        }
      }
    }
    return false;
  }

  Blossom(int n) : // n = vertices
    vis(n + 1), dad(n + 1), orig(n + 1), match(n + 1),
    aux(n + 1), conn(n + 1), t(0), N(n) {
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 15 of 25

```
    for(int i = 0; i <= n; ++i) {
      conn[i].clear();
      match[i] = aux[i] = dad[i] = 0;
    }
  }

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

  int solve() { // call this for answer
    int ans = 0;
    vector<int> V(N - 1);
    iota(V.begin(), V.end(), 1);
    shuffle(V.begin(), V.end(), mt19937(0x94949));
    for(auto x : V) {
      if(!match[x]) {
        for(auto y : conn[x]) {
          if(!match[y]) {
            match[x] = y, match[y] = x;
            ++ans;
            break;
          }
        }
      }
    }
    for(int i = 1; i <= N; ++i) {
      if(!match[i] && bfs(i))
        ++ans;
    }
    return ans;
  }
};
```

5.8 Eulerian Path or Cycle

```
// finds a eulerian path / cycle
// visits each edge only once
// properties:
// - cycle: degrees are even
// - path: degrees are even OR degrees are even except for 2 vertices
// how to use: g = adjacency list g[n] = connected to n, undirected
// if there is a vertex u with an odd degree, call dfs(u)
// else call on any vertex
// ans = path result

vector<set<int>> g;
vector<int> ans;

void dfs(int u) {
  while(g[u].size()) {
    int v = *g[u].begin();
    g[u].erase(v);
    g[v].erase(u);
    dfs(v);
  }
  ans.push_back(u);
}
```

5.9 Hierholzer's Algorithm

```
// Eulerian on Directed Graph
stack<int> path;
vector<int> euler;
inline void hierholzer() {
  path.push(0);
  int cur = 0;
  while(!path.empty()) {
    if(!adj[cur].empty()) {
      path.push(cur);
      int next = adj[cur].back();
      adj[cur].pob();
      cur = next;
    } else {
      euler.pb(cur);
      cur = path.top();
      path.pop();
    }
  }
  reverse(euler.begin(), euler.end());
}
```

5.10 2-SAT

```
struct TwoSAT {
  int n;
  vector<vector<int>> g, gr;
  vector<int> comp, topological_order, answer;
  vector<bool> vis;

  TwoSAT() {}
  TwoSAT(int _n) :
    n(_n), g(2 * n), gr(2 * n), comp(2 * n), answer(2 * n), vis(2 * n) {}

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

  // For the following three functions
  // int x, bool val: if 'val' is true, we take the variable to be x.
  // Otherwise we take it to be x's complement.

  // At least one of them is true
  void add_clause_or(int i, bool f, int j, bool p) {
    add_edge(i + (f ? n : 0), j + (p ? 0 : n));
    add_edge(j + (p ? n : 0), i + (f ? 0 : n));
  }

  // Only one of them is true
  void add_clause_xor(int i, bool f, int j, bool p) {
    add_clause_or(i, f, j, p);
    add_clause_or(i, !f, j, !p);
  }

  // Both of them have the same value
  void add_clause_and(int i, bool f, int j, bool p) {
    add_clause_xor(i, !f, j, p);
  }

  // Topological sort
  void dfs(int u) {
    vis[u] = true;
    for(const auto& v : g[u])
      if(!vis[v])
        dfs(v);
    topological_order.push_back(u);
  }

  // Extracting strongly connected components
  void scc(int u, int id) {
    vis[u] = true;
    comp[u] = id;
    for(const auto& v : gr[u])
      if(!vis[v])
        scc(v, id);
  }
```


Proof by... forgor
Bina Nusantara University

```
  bool satisfiable() {
    fill(vis.begin(), vis.end(), false);
    for(int i = 0; i < 2 * n; i++)
      if(!vis[i])
        dfs(i);
    fill(vis.begin(), vis.end(), false);
    reverse(topological_order.begin(), topological_order.end());
    int id = 0;
    for(const auto& v : topological_order)
      if(!vis[v])
        scc(v, id++);
    // Constructing the answer
    for(int i = 0; i < n; i++) {
      if(comp[i] == comp[i + n])
        return false;
      answer[i] = (comp[i] > comp[i + n] ? 1 : 0);
    }
    return true;
  }
};
```

6 Math

6.1 Extended Euclidean GCD

```
// computes x and y such that ax + by = gcd(a, b) in O(log (min(a, b)))
// returns {gcd(a, b), x, y}
tuple<int, int, int> gcd(int a, int b) {
  if(b == 0) return {a, 1, 0};
  auto [d, x1, y1] = gcd(b, a % b);
  return {d, y1, x1 - y1* (a / b)};
}
```

6.2 Generalized CRT

```
template<typename T>
T extended_euclid(T a, T b, T& x, T& y) {
  if(b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  T xx, yy, gcd;
  gcd = extended_euclid(b, a % b, xx, yy);
  x = yy;
  y = xx - (yy * (a / b));
  return gcd;
}
template<typename T>
T MOD(T a, T b) {
  return (a % b + b) % b;
}
// return x, lcm. x = a % n && x = b % m
template<typename T>
pair<T, T> CRT(T a, T n, T b, T m) {
  T _n, _m;
  T gcd = extended_euclid(n, m, _n, _m);
  if(n == m) {
    if(a == b)
      return pair<T, T>(a, n);
    else
      return pair<T, T>(-1, -1);
  } else if(abs(a - b) % gcd != 0)
    return pair<T, T>(-1, -1);
  else {
    T lcm = m * n / gcd;
    T x = MOD(a + MOD(n * MOD(_n * ((b - a) / gcd), m / gcd), lcm), lcm);
    return pair<T, T>(x, lcm);
  }
}
```

6.3 Generalized Lucas Theorem

```
/*Special Lucas : (n,k) % p^x
  fctp[n] =  Product of the integers less than or equal
  to n that are not divisible by p
  Precompute fctp*/
LL p
LL E(LL n, int m) {
  LL tot = 0;
  while(n != 0)
    tot += n / m, n /= m;
  return tot;
}
LL funct(LL n, LL base) {
  LL ans = fast(fctp[base], n / base, base) * fctp[n % base] % base;
  return ans;
}
LL F(LL n, LL base) {
  LL ans = 1;
  while(n != 0) {
    ans = (ans * funct(n, base)) % base;
    n /= p;
  }
  return ans;
}
LL special_lucas(LL n, LL r, LL base) {
  p = fprime(base);
  LL pow = E(n, p) - E(n - r, p) - E(r, p);
  LL TOP = fast(p, pow, base) * F(n, base) % base;
  LL BOT = F(r, base) * F(n - r, base) % base;
  return (TOP * fast(BOT, totien(base) - 1, base)) % base;
}
//End of Special Lucas
```

6.4 Linear Diophantine

```
//FOR SOLVING MINIMUM ABS(X) + ABS(Y)
ll x, y, newX, newY, target = 0;
ll extGcd(ll a, ll b) {
  if(b == 0) {
    x = 1, y = 0;
    return a;
  }
  ll ret = extGcd(b, a % b);
  newX = y;
  newY = x - y * (a / b);
  x = newX;
  y = newY;
  return ret;
}
ll fix(ll sol, ll rt) {
  ll ret = 0;
  //CASE SOLUTION(X/Y) < TARGET
  if(sol < target)
    ret = -floor(abs(sol + target) / (double)rt);
  //CASE SOLUTION(X/Y) > TARGET
  if(sol > target)
    ret = ceil(abs(sol - target) / (double)rt);
  return ret;
}
ll work(ll a, ll b, ll c) {
  ll gcd = extGcd(a, b);
  ll solX = x * (c / gcd);
  ll solY = y * (c / gcd);
  a /= gcd;
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 17 of 25

```
  b /= gcd;
  ll fi = abs(fix(solX, b));
  ll se = abs(fix(solY, a));
  ll lo = min(fi, se);
  ll hi = max(fi, se);
  ll ans = abs(solX) + abs(solY);
  for(ll i = lo; i <= hi; i++) {
    ans = min(ans, abs(solX + i * b) + abs(solY - i * a));
    ans = min(ans, abs(solX - i * b) + abs(solY + i * a));
  }
  return ans;
}
```

6.5 Modular Linear Equation

```
// finds all solutions to ax = b (mod n)
vi modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  vi ret;
  int g = extended_euclid(a, n, x, y);
  if(!(b % g)) {
    x = mod(x * (b / g), n);
    for(int i = 0; i < g; i++)
      ret.push_back(mod(x + i * (n / g), n));
  }
  return ret;
}
```

6.6 Miller-Rabin and Pollard's Rho

```
namespace MillerRabin {
  const vector<ll> primes = { // deterministic up to 2^64 - 1
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37
  };
  ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
  }
  ll powa(ll x, ll y, ll p) { // (x ^ y) % p
    if(!y)
      return 1;
    if(y & 1)
      return ((__int128) x * powa(x, y - 1, p)) % p;
    ll temp = powa(x, y >> 1, p);
    return ((__int128) temp * temp) % p;
  }
  bool miller_rabin(ll n, ll a, ll d, int s) {
    ll x = powa(a, d, n);
    if(x == 1 || x == n - 1)
      return 0;
    for(int i = 0; i < s; ++i) {
      x = ((__int128) x * x) % n;
      if(x == n - 1)
        return 0;
    }
    return 1;
  }
  bool is_prime(ll x) { // use this
    if(x < 2)
      return 0;
    int r = 0;
    ll d = x - 1;
    while((d & 1) == 0) {
      d >>= 1;
      ++r;
    }
    for(auto& i : primes) {
      if(x == i)
        return 1;
      if(miller_rabin(x, i, d, r))
        return 0;
    }
    return 1;
  }
}

namespace PollardRho {
  mt19937_64 generator(chrono::steady_clock::now()
                       .time_since_epoch().count());
  uniform_int_distribution<ll> rand_ll(0, LLONG_MAX);
  ll f(ll x, ll b, ll n) { // (x^2 + b) % n
    return (((__int128) x * x) % n + b) % n;
  }
  ll rho(ll n) {
    if(n % 2 == 0)
      return 2;
    ll b = rand_ll(generator);
    ll x = rand_ll(generator);
    ll y = x;
    while(1) {
      x = f(x, b, n);
      y = f(f(y, b, n), b, n);
      ll d = MillerRabin::gcd(abs(x - y), n);
      if(d != 1)
        return d;
    }
  }
  void pollard_rho(ll n, vector<ll>& res) {
    if(n == 1)
      return;
    if(MillerRabin::is_prime(n)) {
      res.push_back(n);
      return;
    }
    ll d = rho(n);
    pollard_rho(d, res);
    pollard_rho(n / d, res);
  }
  vector<ll> factorize(ll n, bool sorted = 1) { // use this
    vector<ll> res;
    pollard_rho(n, res);
    if(sorted)
      sort(res.begin(), res.end());
    return res;
  }
}
```

6.7 Berlekamp-Massey

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
typedef long long ll;
#define SZ 233333
const int MOD = 1e9 + 7; //or any prime
ll qp(ll a, ll b) {
  ll x = 1;
  a %= MOD;
  while(b) {
    if(b & 1)
      x = x * a % MOD;
    a = a * a % MOD;
    b >>= 1;
  }
  return x;
}
namespace linear_seq {
  vector<int> BM(vector<int> x) {
    //ls: (shortest) relation sequence (after filling zeroes) so far
    //cur: current relation sequence
```


Proof by... forgor
Bina Nusantara University

Page 18 of 25

```
    vector<int> ls, cur;
    //lf: the position of ls (t')
    //ld: delta of ls (v')
    int lf = -1, ld = -1;
    for(int i = 0; i < int(x.size()); ++i) {
      ll t = 0;
      //evaluate at position i
      for(int j = 0; j < int(cur.size()); ++j)
        t = (t + x[i - j - 1] * (ll)cur[j]) % MOD;
      if((t - x[i]) % MOD == 0) {
        continue;  //good so far
      }
      //first non-zero position
      if(!cur.size()) {
        cur.resize(i + 1);
        lf = i;
        ld = (t - x[i]) % MOD;
        continue;
      }
      //cur=cur-c/ld*(x[i]-t)
      ll k = -(x[i] - t) * qp(ld, MOD - 2) % MOD/*1/ld*/;
      vector<int> c(i - lf - 1); //add zeroes in front
      c.pb(k);
      for(int j = 0; j < int(ls.size()); ++j)
        c.pb(-ls[j]*k % MOD);
      if(c.size() < cur.size())
        c.resize(cur.size());
      for(int j = 0; j < int(cur.size()); ++j)
        c[j] = (c[j] + cur[j]) % MOD;
      //if cur is better than ls, change ls to cur
      if(i - lf + (int)ls.size() >= (int)cur.size())
        ls = cur, lf = i, ld = (t - x[i]) % MOD;
      cur = c;
    }
    for(int i = 0; i < int(cur.size()); ++i)
      cur[i] = (cur[i] % MOD + MOD) % MOD;
    return cur;
  }
  int m; //length of recurrence
//a: first terms
//h: relation
  ll a[SZ], h[SZ], t_[SZ], s[SZ], t[SZ];
//calculate p*q mod f
  void mull(ll* p, ll* q) {
    for(int i = 0; i < m + m; ++i)
      t_[i] = 0;
    for(int i = 0; i < m; ++i) if(p[i])
        for(int j = 0; j < m; ++j)
          t_[i + j] = (t_[i + j] + p[i] * q[j]) % MOD;
    for(int i = m + m - 1; i >= m; --i) if(t_[i])
        //miuns t_[i]x^{i-m}(x^m-\sum_{j=0}^{m-1} x^{m-j-1}h_j)
        for(int j = m - 1; ~j; --j)
          t_[i - j - 1] = (t_[i - j - 1] + t_[i] * h[j]) % MOD;
    for(int i = 0; i < m; ++i)
      p[i] = t_[i];
  }
  ll calc(ll K) {
    for(int i = m; ~i; --i)
      s[i] = t[i] = 0;
    //init
    s[0] = 1;
    if(m != 1)
      t[1] = 1;
    else
      t[0] = h[0];
    //binary-exponentiation
    while(K) {
      if(K & 1)
        mull(s, t);
      mull(t, t);
      K >>= 1;
    }
    ll su = 0;
    for(int i = 0; i < m; ++i)
      su = (su + s[i] * a[i]) % MOD;
    return (su % MOD + MOD) % MOD;
  }
  int work(vector<int> x, ll n) {
    if(n < int(x.size()))
      return x[n];
    vector<int> v = BM(x);
    m = v.size();
    if(!m)
      return 0;
    for(int i = 0; i < m; ++i)
      h[i] = v[i], a[i] = x[i];
    return calc(n);
  }
}
using linear_seq::work;

const vector<int> sequence = {
  0, 2, 2, 28, 60, 836, 2766
};
int main() {
  cout << work(sequence, 7) << '\n';
}
```

6.8 Fast Fourier Transform

```
using ld = double; // change to long double if reach 10^18
using cd = complex<ld>;
const ld PI = acos(-(ld)1);

void fft(vector<cd>& a, int sign = 1) {
  int n = a.size();
  ld theta = sign * 2 * PI / n;
  for(int i = 0, j = 1; j < n - 1; j++) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i)
      swap(a[i], a[j]);
  }
  for(int m, mh = 1; (m = mh << 1) <= n; mh = m) {
    int irev = 0;
    for(int i = 0; i < n; i += m) {
      cd w = exp(cd(0, theta * irev));
      for(int k = n >> 2; k > (irev ^= k); k >>= 1);
      for(int j = i; j < mh + i; j++) {
        int k = j + mh;
        cd x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
      }
    }
  }
  if(sign == -1) for(cd& i : a)
      i /= n;
}

vector<ll> multiply(vector<ll> const& a, vector<ll> const& b) {
  vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  int n = 1;
  while(n < a.size() + b.size())
    n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa);
  fft(fb);
  for(int i = 0; i < n; i++)
    fa[i] *= fb[i];
  fft(fa, -1);
```


Proof by... forgor
Bina Nusantara University

Page 19 of 25

```
  vector<ll> res(n);
  for(int i = 0; i < n; i++)
    res[i] = round(fa[i].real());
  return res;
}
```

6.9 Number Theoretic Transform

```
namespace FFT {
  /* ----- Adjust the constants here ----- */
  const int LN = 24; //23
  const int N = 1 << LN;
  typedef long long LL; // 2**23 * 119 + 1. 998244353
// `MOD` must be of the form 2**`LN` * k + 1, where k odd.
  const LL MOD = 9223372036737335297; // 2**24 * 54975513881 + 1.
  const LL PRIMITIVE_ROOT = 3; // Primitive root modulo `MOD`.
  /* ----- End of constants ----- */
  LL root[N];
  inline LL power(LL x, LL y) {
    LL ret = 1;
    for(; y; y >>= 1) {
      if(y & 1)
        ret = (__int128) ret * x % MOD;
      x = (__int128) x * x % MOD;
    }
    return ret;
  }
  inline void init_fft() {
    const LL UNITY = power(PRIMITIVE_ROOT, MOD - 1 >> LN);
    root[0] = 1;
    for(int i = 1; i < N; i++)
      root[i] = (__int128) UNITY * root[i - 1] % MOD;
    return;
  }
// n = 2^k is the length of polynom
  inline void fft(int n, vector<LL>& a, bool invert) {
    for(int i = 1, j = 0; i < n; ++i) {
      int bit = n >> 1;
      for(; j >= bit; bit >>= 1)
        j -= bit;
      j += bit;
      if(i < j)
        swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1) {
      LL wlen = (invert ? root[N - N / len] : root[N / len]);
      for(int i = 0; i < n; i += len) {
        LL w = 1;
        for(int j = 0; j<len >> 1; j++) {
          LL u = a[i + j];
          LL v = (__int128) a[i + j + len / 2] * w % MOD;
          a[i + j] = ((__int128) u + v) % MOD;
          a[i + j + len / 2] = ((__int128) u - v + MOD) % MOD;
          w = (__int128) w * wlen % MOD;
        }
      }
    }
    if(invert) {
      LL inv = power(n, MOD - 2);
      for(int i = 0; i < n; i++)
        a[i] = (__int128) a[i] * inv % MOD;
    }
    return;
  }
  inline vector<LL> multiply(vector<LL> a, vector<LL> b) {
    vector<LL> c;
    int len = 1 << 32 - __builtin_clz(a.size() + b.size() - 2);
    a.resize(len, 0);
    b.resize(len, 0);
    fft(len, a, false);
    fft(len, b, false);
    c.resize(len);
    for(int i = 0; i < len; ++i)
      c[i] = (__int128) a[i] * b[i] % MOD;
    fft(len, c, true);
    return c;
  }
//FFT::init_fft(); wajib di panggil init di awal
}
```

6.10 Gauss-Jordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]
const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;
T GaussJordan(VVT& a, VVT& b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;
  for(int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for(int j = 0; j < n; j++) if(!ipiv[j])
        for(int k = 0; k < n; k++) if(!ipiv[k])
            if(pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) {
              pj = j;
              pk = k;
            }
    if(fabs(a[pj][pk]) < EPS) {
      cerr << "Matrix is singular." << endl;
      exit(0);
    }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if(pj != pk)
      det *= -1;
    irow[i] = pj;
    icol[i] = pk;
    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for(int p = 0; p < n; p++)
      a[pk][p] *= c;
    for(int p = 0; p < m; p++)
      b[pk][p] *= c;
    for(int p = 0; p < n; p++) if(p != pk) {
        c = a[p][pk];
        a[p][pk] = 0;
        for(int q = 0; q < n; q++)
          a[p][q] -= a[pk][q] * c;
        for(int q = 0; q < m; q++)
```

```
            b[p][q] -= b[pk][q] * c;
        }
    }
    for(int p = n - 1; p >= 0; p--) if(irow[p] != icol[p]) {
        for(int k = 0; k < n; k++)
          swap(a[k][irow[p]], a[k][icol[p]]);
      }
    return det;
}
int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1, 2, 3, 4}, {1, 0, 1, 0}, {5, 3, 2, 4}, {6, 1, 4, 6} };
  double B[n][m] = { {1, 2}, {4, 3}, {5, 6}, {8, 7} };
  VVT a(n), b(n);
  for(int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }
  double det = GaussJordan(a, b);
  // expected: 60
  cout << "Determinant: " << det << endl;
  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }
  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for(int i = 0; i < n; i++) {
    for(int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

6.11 Derangement

```
der[0] = 1;
der[1] = 0;
for(int i = 2; i <= 10; ++i)
  der[i] = (ll)(i - 1) * (der[i - 1] + der[i - 2]);
```

6.12 Bernoulli Number

$\sum_{k=1}^{n} k^m = \frac{1}{m+1} \sum_{i=0}^{m} \binom{m+1}{i} B_i^+ n^{m+1-i} = m! \sum_{i=0}^{m} \frac{B_i^+ n^{m+1-i}}{i!(m+1-i)!}$

$B_n^+ = 1 - \sum_{i=0}^{n-1} \binom{n}{i} \frac{B_i^+}{n-i+1}, \quad B_0^+ = 1$

6.13 Forbenius Number

(X* Y) - (X + Y) and total count is(X - 1)* (Y - 1) / 2

6.14 Stars and Bars with Upper Bound

$P = (1 - X^{r_1+1}) \dots (1 - X^{r_n+1}) = \sum_i c_i X^{e_i}$

$Ans = \sum_i c_i \binom{N-e_i+n-1}{n-1}$

6.15 Arithmetic Sequences

$U_n = a + (n-1)a_1 + \frac{(n-1)(n-2)}{1\times2} a_2 + \ldots + \frac{(n-1)(n-2)(n-3)\ldots}{1\times2\times3\times\ldots} a_r$

$S_n = n \times a + \frac{n(n-1)}{1\times2} a_1 + \frac{n(n-1)(n-2)}{1\times2\times3} a_2 + \ldots + \frac{n(n-1)(n-2)(n-3)\ldots}{1\times2\times3\ldots} a_r$

6.16 FWHT

```
// Desc  : Transform a polynom to obtain a_i * b_j * x^(i XOR j) or combinations
// Time  : O(N log N) with N = 2^K
// OP  => c00 c01 c10 c11 | c00 c01 c10 c11 inv
// XOR =>  +1  +1  +1  -1 |  +1  +1  +1  -1 | div the inverse with size = n
// AND =>   1  +1   0   1 |   1  -1   0   1 | no comment
//  OR =>   1   0  +1   1 |   1   0  -1   1 | no comment
typedef vector<long long> vec;
void FWHT(vec& a) {
  int n = a.size();
  for(int lvl = 1; 2 * lvl <= n; lvl <<= 1) {
    for(int i = 0; i < n; i += 2 * lvl) {
      for(int j = 0; j < lvl; j++) { // do not forget to modulo
        long long u = a[i + j], v = a[i + lvl + j];
        a[i + j]       = u + v; // c00 * u + c01 * v
        a[i + lvl + j] = u - v; // c10 * u + c11 * v
      }
    }
  }
}  // you can convolve as usual
```

6.17 Basis Vector

```
int basis[d]; // basis[i] keeps the mask of the vector whose f value is i
int sz;       // Current size of the basis
void insertVector(int mask) {
  for(int i = 0; i < d; i++) {
    if((mask & 1 << i) == 0) {
      continue; // continue if i != f(mask)
    }
    if(!basis[i]) {  // If there is no basis vector with the i'th bit set,
      // then insert this vector into the basis
      basis[i] = mask;
      ++sz;
      return;
    }
    mask ^= basis[i]; // Otherwise subtract the basis vector from this
    // vector
  }
}
```

7 Strings

7.1 Aho-Corasick

```
const int K = 26;
struct Vertex {
  int next[K];
  bool leaf = 0;
  int p = -1, ans = 0;
  char pch;
  int link = -1, mlink = -1;
  //magic link, is the link to find the nearest leaf
  int go[K];
  Vertex(int p = -1, char ch = '$') : p(p), pch(ch) {
    fill(begin(next), end(next), -1);
    fill(begin(go), end(go), -1);
  }
};
vector<Vertex> t;
```

```
int add_string(string const& s) {
  int v = 0;
  for(char ch : s) {
    int c = ch - 'a';
    if(t[v].next[c] == -1) {
      t[v].next[c] = t.size();
      t.emplace_back(v, ch);
    }
    v = t[v].next[c];
  }
  t[v].leaf = 1;
  return v;
}
int go(int v, char ch);
int get_link(int v) {
  if(t[v].link == -1) {
    if(v == 0 || t[v].p == 0)
      t[v].link = 0;
    else
      t[v].link = go(get_link(t[v].p), t[v].pch);
  }
  return t[v].link;
}
int get_mlink(int v) {
  if(t[v].mlink == -1) {
    if(v == 0 || t[v].p == 0)
      t[v].mlink = 0;
    else {
      t[v].mlink = go(get_link(t[v].p), t[v].pch);
      if(t[v].mlink && !t[t[v].mlink].leaf) {
        if(t[t[v].mlink].mlink == -1)
          get_mlink(t[v].mlink);
        t[v].mlink = t[t[v].mlink].mlink;
      }
    }
  }
  return t[v].mlink;
}
int go(int v, char ch) {
  int c = ch - 'a';
  if(t[v].go[c] == -1) {
    if(t[v].next[c] != -1)
      t[v].go[c] = t[v].next[c];
    else
      t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
  }
  return t[v].go[c];
}
//t.pb(Vertex());
```

7.2 Eertree

```
/*
   Eertree - keep track of all palindromes and its occurences
   This code refers to problem Longest Palindromic Substring
https://www.spoj.com/problems/LPS/
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

struct node {
  int next[26];
  int sufflink;
  int len, cnt;
};

const int N = 1e5 + 69;
int n;
string s;
node tree[N];
int idx, suff;
int ans = 0;

void init_eertree() {
  idx = suff = 2;
  tree[1].len = -1, tree[1].sufflink = 1;
  tree[2].len = 0, tree[2].sufflink = 1;
}

bool add_letter(int x) {
  int cur = suff, curlen = 0;
  int nw = s[x] - 'a';
  while(1) {
    curlen = tree[cur].len;
    if(x - curlen - 1 >= 0 && s[x - curlen - 1] == s[x])
      break;
    cur = tree[cur].sufflink;
  }
  if(tree[cur].next[nw]) {
    suff = tree[cur].next[nw];
    return 0;
  }
  tree[cur].next[nw] = suff = ++idx;
  tree[idx].len = tree[cur].len + 2;
  ans = max(ans, tree[idx].len);
  if(tree[idx].len == 1) {
    tree[idx].sufflink = 2;
    tree[idx].cnt = 1;
    return 1;
  }
  while(1) {
    cur = tree[cur].sufflink;
    curlen = tree[cur].len;
    if(x - curlen - 1 >= 0 && s[x - curlen - 1] == s[x]) {
      tree[idx].sufflink = tree[cur].next[nw];
      break;
    }
  }
  tree[idx].cnt = tree[tree[idx].sufflink].cnt + 1;
  return 1;
}

int main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cin >> n >> s;
  init_eertree();
  for(int i = 0; i < n; i++)
    add_letter(i);
  cout << ans << '\n';
  return 0;
}
```

7.3 Manacher's Algorithm

```
// Computes lps array. lps[i] means the longest palindromic substring centered at i (↩
     when i is even, it is between characters. when it is odd, it is on characters)lps↩
     [0] = 0; lps[1] = 1;
REP(i, 2, 2 * str.size()) {
  int l = i / 2 - lps[i] / 2;
  int r = (i - 1) / 2 + lps[i] / 2;
  while(1) { // widen
    if(l == 0 || r + 1 == str.size())
      break;
    if(str[l - 1] != str[r + 1])
      break;
    l--, r++;
  }
  lps[i] = r - l + 1;
```


Page 22 of 25

```
  // jump
  if(lps[i] > 2) {
    int j = i - 1, k = i + 1; // while lps[j] inside lps[i]
    while(lps[j] - j < lps[i] - i)
      lps[k++] = lps[j--];
    lps[k] = lps[i] - (i - j); // set lps[k] to edge of lps[i]
    i = k - 1; // jump to mirror, which is k
  }
}
```

7.4 Suffix Array

```
struct SuffixArray {
  string s;
  vector<int> p, c, lcp;
  int n;
  SuffixArray(string _s) : s(_s) {
    s += '$';
    n = (int)s.size();
    p.resize(n);
    c.resize(n);
    {
      // calculate for k = 0
      vector<pair<char, int>> v(n);
      for(int i = 0; i < n; ++i)
        v[i] = make_pair(s[i], i);
      sort(all(v));
      for(int i = 0; i < n; ++i)
        p[i] = v[i].se;
      c[p[0]] = 0;
      for(int i = 1; i < n; ++i)
        c[p[i]] = c[p[i - 1]] + (v[i].fi != v[i - 1].fi);
    }
    const auto countingSort = [](vector<int>& p, vector<int>& c) -> void {
      int n = (int)p.size();
      vector<int> cnt(n), pos(n);
      for(auto& i : c)
        ++cnt[i];
      for(int i = 1; i < n; ++i)
        pos[i] = pos[i - 1] + cnt[i - 1];
      vector<int> pNew(n);
      for(auto& i : p)
        pNew[pos[c[i]]++] = i;
      p = pNew;
    };
    for(int k = 0; (1 << k) < n; ++k) {
      for(int i = 0; i < n; ++i) {  // transition k -> k + 1
        // shift p[i] by 2^k to the left, so that second elements are
        // sorted
        p[i] = (p[i] + n - (1 << k)) % n;
      }
      countingSort(p, c);
      vector<int> cNew(n);
      for(int i = 1; i < n; ++i) {
        pair<int, int> cur = make_pair(
                               c[p[i]], c[(p[i] + (1 << k)) % n]
                             );
        pair<int, int> pre = make_pair(
                               c[p[i - 1]], c[(p[i - 1] + (1 << k)) % n]
                             );
        cNew[p[i]] = cNew[p[i - 1]] + (cur != pre);
      }
      c = cNew;
    }
    // lcp[i]: longest common prefix of s[p[i]] and s[p[i-1]]
    lcp.resize(n); // iterate from the longest suffix
    for(int i = 0, k = 0; i + 1 < n; ++i) {
      int pi = c[i]; // rank of suffix [i..]
      int j = p[pi - 1];
      for(; s[i + k] == s[j + k]; ++k)
        ;
      lcp[pi] = k;
      k = max(k - 1, 0);
    }
  }
};
```

7.5 Suffix Automaton

```
struct state {
  int len, link;
  map<char, int>next; //use array if TLE
};

const int MAXLEN = 100005;
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
  sz = last = 0;
  st[0].len = 0;
  st[0].link = -1;
  st[0].next.clear();
  ++sz;
}
void sa_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  st[cur].next.clear();
  int p;
  for(p = last; p != -1 && !st[p].next.count(c); p = st[p].link)
    st[p].next[c] = cur;
  if(p == -1)
    st[cur].link = 0;
  else {
    int q = st[p].next[c];
    if(st[p].len + 1 == st[q].len)
      st[cur].link = q;
    else {
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;
      for(; p != -1 && st[p].next[c] == q; p = st[p].link)
        st[p].next[c] = clone;
      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}
// forwarding
for(int i = 0; i < m; i++) {
  while(cur >= 0 && st[cur].next.count(pa[i]) == 0) {
    cur = st[cur].link;
    if(cur != -1)
      len = st[cur].len;
  }
  if(st[cur].next.count(pa[i])) {
    len++;
    cur = st[cur].next[pa[i]];
  } else
    len = cur = 0;
}
// shortening abc -> bc
if(l == m) {
  l--;
  if(l <= st[st[cur].link].len)
    cur = st[cur].link;
}
// finding lowest and highest length
```


Proof by... forgor
Bina Nusantara University
Owmicron, ZeroScar, KerakTelor
Page 23 of 25

Owen Djonatan, Luis Anthonie Alkins, Richard Alison

```
int lo = st[st[cur].link].len + 1;
int hi = st[cur].len;
//Finding number of distinct substrings
//answer = distsub(0)
LL d[MAXLEN * 2];
LL distsub(int ver) {
  LL  tp = 1;
  if(d[ver])
    return d[ver];
  for(map<char, int>::iterator it = st[ver].next.begin();
      it != st[ver].next.end(); it++)
    tp += distsub(it->second);
  d[ver] = tp;
  return d[ver];
}
//Total Length of all distinct substrings
//call distsub first before call lesub
LL ans[MAXLEN * 2];
LL lesub(int ver) {
  LL tp = 0;
  if(ans[ver])
    return ans[ver];
  for(map<char, int>::iterator it = st[ver].next.begin();
      it != st[ver].next.end(); it++)
    tp += lesub(it->second) + d[it->second];
  ans[ver] = tp;
  return ans[ver];
}
//find the k-th lexicographical substring
void kthsub(int ver, int K, string& ret) {
  for(map<char, int>::iterator it = st[ver].next.begin();
      it != st[ver].next.end(); it++) {
    int v = it->second;
    if(K <= d[v]) {
      K--;
      if(K == 0) {
        ret.push_back(it->first);
        return;
      } else {
        ret.push_back(it->first);
        kthsub(v, K, ret);
        return;
      }
    } else
      K -= d[v];
  }
}
// Smallest Cyclic Shift to obtain lexicographical smallest of All possible
//in int main do this
int main() {
  string S;
  sa_init();
  cin >> S; //input
  tp = 0;
  t = S.length();
  S += S;
  for(int a = 0; a < S.size(); a++)
    sa_extend(S[a]);
  minshift(0);
}
//the function
int tp, t;
void minshift(int ver) {
  for(map<char, int>::iterator it = st[ver].next.begin();
      it != st[ver].next.end(); it++) {
    tp++;
    if(tp == t) {
      cout << st[ver].len - t + 1 << endl;
      break;
    }
    minshift(it->second);
    break;
  }
}
//end of function
// LONGEST COMMON SUBSTRING OF TWO STRINGS
string lcs(string s, string t) {
  sa_init();
  for(int i = 0; i < (int)s.length(); ++i)
    sa_extend(s[i]);
  int v = 0,  l = 0,
      best = 0,  bestpos = 0;
  for(int i = 0; i < (int)t.length(); ++i) {
    while(v && ! st[v].next.count(t[i])) {
      v = st[v].link;
      l = st[v].length;
    }
    if(st[v].next.count(t[i])) {
      v = st[v].next[t[i]];
      ++l;
    }
    if(l > best)
      best = l,  bestpos = i;
  }
  return t.substr(bestpos - best + 1, best);
}
```

7.6 KMP

```
auto get_kmp = [&](string S, string T) -> vector<int> {
  // S is the text, T is the pattern // ababa aba -> expected: {1 2 3 2 3}
  int N = S.size(), M = T.size();
  vector<int> lps(M), kmp(N);
  for(int i = 1, j = 0; i < M;) {
    if(T[i] == T[j])
      lps[i++] = ++j;
    else {
      if(j)
        j = lps[j - 1];
      else
        lps[i++] = 0;
    }
  }
  for(int i = 0, j = 0; i < N;) {
    if(S[i] == T[j]) {
      kmp[i++] = ++j;
      if(j == M)
        j = lps[j - 1];
    } else {
      if(j)
        j = lps[j - 1];
      else
        kmp[i++] = 0;
    }
  }
  return kmp;
};
```

8 OEIS

8.1 A000108 (Catalan)

```
Catalan numbers
f(n) = nCk(2n,n) / (n+1) = nCk(2n,n) - nCk(2n,n+1) = f(n-1) * 2*(2*n-1) / (n+1)
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900,
2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420,
24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452,
18367353072152, 69533550916004, 263747951750360, 1002242216651368,
3814986502092304
```

8.2 A018819

```
Binary partition function: number of partitions of n into powers of 2
f(2m+1) = f(2m);  f(2m) = f(2m-1) + f(m)
1, 1, 2, 2, 4, 4, 6, 6, 10, 10, 14, 14, 20, 20, 26, 26, 36, 36, 46, 46, 60,
60, 74, 74, 94, 94, 114, 114, 140, 140, 166, 166, 202, 202, 238, 238, 284,
284, 330, 330, 390, 390, 450, 450, 524, 524, 598, 598, 692, 692, 786, 786,
900, 900, 1014, 1014, 1154, 1154, 1294, 1294
```

8.3 A092098

```
3-Portolan numbers: number of regions formed by n-secting the angles of
an equilateral triangle.
long long solve(long long n) {
    long long res = (n % 2 == 1 ? 3*n*n - 3*n + 1 : 3*n*n - 6*n + 6);
    const int bats = n/2 - 1;
    for (long long i=1; i<=bats; i++) for (long long j=1; j<=bats; j++) {
        long long num = i * (n-j) * n;
        long long denum = (n-i) * j + i * (n-j);
        res -= 6 * (num % denum == 0 && num / denum <= bats);
    } return res;
}
1, 6, 19, 30, 61, 78, 127, 150, 217, 246, 331, 366, 469, 510, 625, 678, 817,
870, 1027, 1080, 1261, 1326, 1519, 1566, 1801, 1878, 2107, 2190, 2437, 2520,
2791, 2886, 3169, 3270, 3559, 3678, 3997, 4110, 4447, 4548, 4921, 5034, 5419,
5550, 5899, 6078, 6487
```

8.4 A000127

```
Maximal number of regions obtained by joining n points around a circle by
straight lines
f(n) = (n^4 - 6*n^3 + 23*n^2 - 18*n + 24) / 24
1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, 562, 794, 1093, 1471, 1941, 2517,
3214, 4048, 5036, 6196, 7547, 9109, 10903, 12951, 15276, 17902, 20854, 24158,
27841, 31931, 36457, 41449, 46938, 52956, 59536, 66712, 74519, 82993, 92171,
102091, 112792, 124314
```

8.5 A001534

```
Number of graphs with n nodes and n edges.
0, 0, 1, 2, 6, 21, 65, 221, 771, 2769, 10250, 39243, 154658, 628635, 2632420,
11353457, 50411413, 230341716, 1082481189, 5228952960, 25945377057,
132140242356, 690238318754
```


Proof by... forgor
Bina Nusantara University

Page 25 of 25