Contents

1 Build and Snippet

1.1 Sublime Build

```
#for linux
  {
    "shell_cmd": "g++ $file -o $file_base_name && ./
        $file_base_name<input.txt> output.txt && rm
        $file_base_name",
    "working_dir": "$file_path",
    "selector": "source.c++"
  }
#for windows
  {
  "shell_cmd": "g++ -std=c++17 $file -o $file_base_name.
      exe && $file_base_name.exe <input.txt> output.txt
      && del $file_base_name.exe",
  "working_dir":"$file_path",
  "selector":"source.c++"
  }
```

2 Data Structures

2.1 2D BIT

```
const int N = 1008;
int bit[N][N], a[N][N], n, m, q;
void update(int x, int y, int val) {
  for (; x < N; x += -x & x)
    for (int j = y; j < N; j += -j & j) bit[x][j] += val;
}
int get(int x, int y) {
  int ans = 0;
  for (; x; x -= x & -x)
    for (int j = y; j; j -= j & -j) ans += bit[x][j];
  return ans;
}
int get(int x1, int y1, int x2, int y2) {
  return get(x2, y2) - get(x1 - 1, y2) - get(x2, y1 - 1)
      + get(x1 - 1, y1 - 1);
}
```

2.2 BIT

```
class BIT {
  int *bin, N;

 public:
  BIT(int N) : N(N) {
    bin = new int[N + 1];
    memset(bin, 0, (N + 1) * sizeof(int));
  }
  void update(int id, int val) {
    for (; id <= N; id += id & -id) bin[id] += val;
  }
  int helper(int id) {
    int sum = 0;
    for (; id > 0; id -= id & -id) sum += bin[id];
    return sum;
  }
  int query(int l, int r) { return helper(r) - helper(l -
      1); }
  ~BIT() { delete[] bin; }
};
```

2.3 Lazy Propagation

```
template <typename node, typename change>
class SegmentTree {
 public:
  int n;

  node *tree, identity;
  node (*merge)(node, node);

  change *lazy, noUpdate;
  void (*applyUpdate)(int, int, node &, change);
  void (*mergeUpdate)(int, int, change &, change);

  void build(vector<node> &input, int lo, int hi, int
      root = 0) {
    if (lo == hi) {
      tree[root] = input[lo];
      return;
    }

    int mid = lo + hi >> 1, leftChild = 2 * root + 1,
        rightChild = 2 * root + 2;
    build(input, lo, mid, leftChild);
    build(input, mid + 1, hi, rightChild);

    tree[root] = merge(tree[leftChild], tree[rightChild])
        ;
  }

  void propagate(int lo, int hi, int root) {
    applyUpdate(lo, hi, tree[root], lazy[root]);
    if (lo < hi) {
      int mid = lo + hi >> 1, leftChild = 2 * root + 1,
          rightChild = 2 * root + 2;

      mergeUpdate(lo, mid, lazy[leftChild], lazy[root]);
      mergeUpdate(mid + 1, hi, lazy[rightChild], lazy[
          root]);
    }

    lazy[root] = noUpdate;
  }

  void update(int from, int to, int lo, int hi, int root,
      change delta) {
    if (lo > hi) return;

    propagate(lo, hi, root);
    if (from > hi || to < lo) return;

    if (from <= lo && to >= hi) {
      mergeUpdate(lo, hi, lazy[root], delta);
      propagate(lo, hi, root);
      return;
    }

    int mid = lo + hi >> 1, leftChild = 2 * root + 1,
        rightChild = 2 * root + 2;
    update(from, to, lo, mid, leftChild, delta);
    update(from, to, mid + 1, hi, rightChild, delta);

    tree[root] = merge(tree[leftChild], tree[rightChild])
        ;
  }

  node query(int from, int to, int lo, int hi, int root)
      {
    if (lo > hi) return identity;

    propagate(lo, hi, root);
    if (from > hi || to < lo) return identity;

    if (from <= lo && to >= hi) return tree[root];

    int mid = lo + hi >> 1, leftChild = 2 * root + 1,
        rightChild = 2 * root + 2;
```

```
  node q1 = query(from, to, lo, mid, leftChild), q2 =
      query(from, to, mid + 1, hi, rightChild);
  return merge(q1, q2);
}

int lowerbound(int lo, int hi, int root, node val) {
  if (lo > hi) return -1;
  propagate(lo, hi, root);

  if (tree[root] < val) return -1;
  if (lo == hi) {
    if (tree[root] == val) return hi;
    return -1;
  }

  int mid = lo + hi >> 1, leftChild = 2 * root + 1,
      rightChild = 2 * root + 2;
  int leftSum = query(lo, mid, lo, mid, leftChild);

  if (leftSum >= val)
    return lowerbound(lo, mid, leftChild, val);
  else
    return lowerbound(mid + 1, hi, rightChild, val -
        leftSum);
  // val-leftsum works when merge function is sum
}

// leftmost position of a minsegtree
// that has value <= val
int walk(int lo, int hi, int root, int from, node val)
    {
  if (lo > hi) return -1;

  propagate(lo, hi, root);
  if (from > hi) return -1;
  if (tree[root] > val) return hi;

  if (lo == hi) {
    return hi;
  }

  int mid = lo + hi >> 1, leftChild = 2 * root + 1,
      rightChild = 2 * root + 2;
  if (from > mid) return walk(mid + 1, hi, rightChild,
      from, val);
  node q1 = query(max(from, lo), mid, lo, mid,
      leftChild);

  if (q1 <= val) return walk(lo, mid, leftChild, from,
      val);
  return walk(mid + 1, hi, rightChild, from, val);
}

public:
 SegmentTree(SegmentTree &st) : tree(st.tree), lazy(st.
     lazy), n(st.n), merge(st.merge), identity(st.
     identity), applyUpdate(st.applyUpdate), mergeUpdate
     (st.mergeUpdate), noUpdate(st.noUpdate) {}

 SegmentTree(vector<node> &input, node (*merge)(node,
     node), node identity, void (*applyUpdate)(int, int,
      node &, change), void (*mergeUpdate)(int, int,
     change &, change), change noUpdate)
    : n(input.size()), merge(merge), identity(identity)
        , applyUpdate(applyUpdate), mergeUpdate(
        mergeUpdate), noUpdate(noUpdate) {
  tree = new node[n << 2];
  build(input, 0, n - 1);

  lazy = new change[n << 2];
  fill(lazy, lazy + (n << 2), noUpdate);
 }

 node query(int from, int to) {
  if (from > to || to > n) return identity;
  return query(from, to, 0, n - 1, 0);
 }

 void update(int from, int to, change delta) { update(
     from, to, 0, n - 1, 0, delta); }

 int lowerbound(int val)
 // Only works for non-decreasing function
 {
  return lowerbound(0, n - 1, 0, val);
 }

 ~SegmentTree() {
  delete[] tree;
  delete[] lazy;
 }
};

int merge(int a, int b) { return a + b; }

void applyUpdate(int lo, int hi, int &val, int delta) {
    val += delta * (hi - lo + 1); }

void mergeUpdate(int lo, int hi, int &val, int delta) {
    val += delta; }
```

2.4 MergeSortTree

```
class MergeSortTree {
  int n;
  vector<vector<int>> tree;
  void build(int id, int le, int ri, vector<int> &a) {
    if (le == ri) {
      tree[id].push_back(a[le]);
      return;
    }
    int mid = (le + ri) >> 1;
    build(2 * id + 1, le, mid, a);
    build(2 * id + 2, mid + 1, ri, a);

    auto &left = tree[2 * id + 1], &right = tree[2 * id +
        2];
    int i = 0, j = 0, n = left.size(), m = right.size();
    while (i < n && j < m) {
      if (left[i] < right[j])
        tree[id].push_back(left[i]), i++;
      else
        tree[id].push_back(right[j]), j++;
    }
    while (i < n) tree[id].push_back(left[i]), i++;
    while (j < m) tree[id].push_back(right[j]), j++;
  }

  // number of element greater than val
  int queryL(int id, int le, int ri, int l, int r, int
      val) {
    if (le > r || ri < l) {
      return 0;
    }
    if (le >= l && ri <= r) {
      return ri - le + 1 - (upper_bound(tree[id].begin(),
          tree[id].end(), val) - tree[id].begin());
```

```
    }
    int mid = (le + ri) >> 1;
    return queryL(2 * id + 1, le, mid, l, r, val) +
        queryL(2 * id + 2, mid + 1, ri, l, r, val);
  }

  // number of element smaller than val
  int queryS(int id, int le, int ri, int l, int r, int
    val) {
    if (le > r || ri < l) {
      return 0;
    }

    if (le >= l && ri <= r) {
      return (upper_bound(tree[id].begin(), tree[id].end
        (), val - 1) - tree[id].begin());
    }

    int mid = (le + ri) >> 1;
    return queryS(2 * id + 1, le, mid, l, r, val) +
        queryS(2 * id + 2, mid + 1, ri, l, r, val);
  }

 public:
  MergeSortTree(vector<int> &a) {
    n = a.size();
    tree.resize(n * 4);
    build(0, 0, n - 1, a);
  }
  int queryS(int l, int r, int val) { return queryS(0, 0,
     n - 1, l, r, val); }
  int queryL(int l, int r, int val) { return queryL(0, 0,
     n - 1, l, r, val); }
};
```

2.5 PST

```
// this calculates xor/xor_hash of all the element less
    than 'x' in [0, i]. query is a walk function

class PST {
#define lc(u) tree[u].left
#define rc(u) tree[u].right;
  struct node {
    int left = 0, right = 0, val = 0;
  };
  node *tree;
  int N, LG, time = 0, I = 0;

  node create(int l, int r) { return {l, r, merge(tree[l
    ].val, tree[r].val)}; }
  int merge(LL a, LL b) { return a ^ b; }
  int build(int le, int ri) {
    int id = ++time;
    if (le == ri) return tree[id] = node(), id;
    int m = (le + ri) / 2;
    return tree[id] = create(build(le, m), build(m + 1,
      ri)), id;
  }
  int update(int id, int le, int ri, int pos, int val) {
    int nid = ++time;
    if (le == ri) return tree[nid] = {0, 0, (tree[id].val
      ^ val)}, nid; // change here
    int m = (le + ri) / 2;
    if (pos <= m) {
      tree[nid] = create(update(tree[id].left, le, m, pos
        , val), tree[id].right);
    } else {
      tree[nid] = create(tree[id].left, update(tree[id].
        right, m + 1, ri, pos, val));
    }
    return nid;
  }
  int query(int id, int di, int le, int ri) {
    if (tree[id].val == tree[di].val) return 0;
    if (le == ri) return le;
    int m = (le + ri) >> 1;
    if (tree[tree[id].left].val != tree[tree[di].left].
      val) return query(tree[id].left, tree[di].left,
      le, m);
    return query(tree[id].right, tree[di].right, m + 1,
      ri);
  }

 public:
  PST(int N, int U) { // U --> number of expected updates
    this->N = N;
    LG = 33 - __builtin_clz(N);
    tree = new node[(N + U) * LG];
    build(0, N - 1);
  }
  int update(int id, int pos, int val) { return update(id
    , 0, N - 1, pos, val); }
  int query(int id, int di) { return query(id, di, 0, N -
     1); }
  ~PST() { delete[] tree; }
};
```

2.6 Segment Tree

```
template <typename DT>
class segmentTree {
  DT *seg, I;
  int n;
  DT (*merge)(DT, DT);

  void build(int idx, int le, int ri, vector<DT> &v) {
    if (le == ri) {
      seg[idx] = v[le];
      return;
    }
    int mid = (le + ri) >> 1;
    build(2 * idx + 1, le, mid, v);
    build(2 * idx + 2, mid + 1, ri, v);
    seg[idx] = merge(seg[2 * idx + 1], seg[2 * idx + 2]);
  }

  void update(int idx, int le, int ri, int pos, DT val) {
    if (le == ri) {
      seg[idx] = val;
      return;
    }
    int mid = (le + ri) >> 1;
    if (pos <= mid)
      update(2 * idx + 1, le, mid, pos, val);
    else
      update(2 * idx + 2, mid + 1, ri, pos, val);
    seg[idx] = merge(seg[2 * idx + 1], seg[2 * idx + 2]);
  }

  DT query(int idx, int le, int ri, int l, int r) {
    if (l <= le && r >= ri) {
      return seg[idx];
    }
    if (r < le || l > ri) {
      return I;
    }
    int mid = (le + ri) >> 1;
    return merge(query(2 * idx + 1, le, mid, l, r), query
      (2 * idx + 2, mid + 1, ri, l, r));
```

```cpp
  }

  // finding the leftmost appearence of value <= val in [
      l....r] range
  // need minimum segment tree
  int walk(int idx, int le, int ri, int l, int r, DT val)
       {
    if (r < le || l > ri) {
      return r;
    }
    if (le == ri) {
      if (seg[idx] <= val) return le;
      return r;
    }
    if (l <= le && r >= ri) {
      int mid = (le + ri) >> 1;
      if (seg[2 * idx + 1] <= val) return walk(2 * idx +
          1, le, mid, l, r, val);
      return walk(2 * idx + 2, mid + 1, ri, l, r, val);
    }
    int mid = (le + ri) >> 1;
    return merge(walk(2 * idx + 1, le, mid, l, r, val),
        walk(2 * idx + 2, mid + 1, ri, l, r, val));
  }

 public:
  segmentTree(vector<DT> &v, DT (*fptr)(DT, DT), DT _I) {
    n = v.size();
    I = _I;
    merge = fptr;
    seg = new DT[4 * n];
    build(0, 0, n - 1, v);
  }
  void update(int pos, int val) { update(0, 0, n - 1, pos
      , val); }
  int walk(int l, int r, DT val) {
    if (query(l, r) > val) return r;
    return walk(0, 0, n - 1, l, r, val);
  }
  DT query(int l, int r) { return query(0, 0, n - 1, l, r
      ); }
};

int fun(int a, int b) { return max(a, b); }
```

2.7 Sparse Table

```cpp
class SparseTable {
 private:
  vector<vector<int>> table;
  vector<int> log;
  int n;

 public:
  SparseTable(const vector<int>& arr) {
    n = arr.size();
    log.resize(n + 1);
    buildLog();
    table = vector<vector<int>>(n, vector<int>(log[n] +
        1));
    for (int i = 0; i < n; i++) {
      table[i][0] = arr[i];
    }
    for (int j = 1; (1 << j) <= n; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        table[i][j] = merge(table[i][j - 1], table[i + (1
             << (j - 1))][j - 1]);
      }
    }
  }
  int merge(int a, int b) { return max(a, b); }
  void buildLog() {
    log[1] = 0;
    for (int i = 2; i <= n; i++)
      log[i] = log[i / 2] + 1;
  }
  int Query(int L, int R) {
    int j = log[R - L + 1];
    return merge(table[L][j], table[R - (1 << j) + 1][j])
        ;
  }
  int query(int L, int R) {
    int sum = 0;
    for (int j = log[R - L + 1]; L <= R; j = log[R - L +
        1]) {
      sum = merge(sum, table[L][j]);
      L += (1 << j);
    }
    return sum;
  }
};
```

2.8 Trie

```cpp
namespace tri {
const int k = 26;
struct trie_vertex {
  int next[k], cPrefix = 0;
  bool leaf = false;
  trie_vertex() { fill(begin(next), end(next), -1); }
};
struct Trie {
  vector<trie_vertex> trie;
  Trie() { trie.resize(1); }
  void insert(string const& s) {
    int v = 0;
    for (char ch : s) {
      int c = ch - 'a';
      if (trie[v].next[c] == -1) {
        trie[v].next[c] = trie.size();
        trie.emplace_back();
      }
      v = trie[v].next[c];
      trie[v].cPrefix++;
    }
    trie[v].leaf = true;
  }
  int search(string const& key, bool& isWord) {
    int v = 0, level = 0;
    for (char ch : key) {
      int c = ch - 'a';
      v = trie[v].next[c];
      if (v == -1) return -1;
      isWord = trie[v].leaf;
      level++;
    }
    return level;
  }
};
}  // namespace tri
```

2.9 sparse table 2D

```cpp
// rectangle query
namespace st2 {
  const int N = 2e3 + 5, B = 12;
  using Ti = long long;
  Ti Id = LLONG_MAX;
  Ti f(Ti a, Ti b) { return max(a, b); }
  Ti tbl[N][N][B];
  void init(int n, int m) {
```

```cpp
    for(int k = 1; k < B; k++) {
     for(int i = 0; i + (1 << k) - 1 < n; i++) {
      for(int j = 0; j + (1 << k) - 1 < m; j++) {
       tbl[i][j][k] = tbl[i][j][k - 1];
       tbl[i][j][k] = f(tbl[i][j][k], tbl[i][j + (1 << k
           - 1)][k - 1]);
       tbl[i][j][k] = f(tbl[i][j][k], tbl[i + (1 << k -
           1)][j][k - 1]);
       tbl[i][j][k] = f(tbl[i][j][k], tbl[i + (1 << k -
           1)][j + (1 << k - 1)][k - 1]);
    } } }
  }
  Ti query(int i, int j, int len) {
    int k = __lg(len);
    LL ret = tbl[i][j][k];
    ret = f(ret, tbl[i + len - (1 << k)][j][k]);
    ret = f(ret, tbl[i][j + len - (1 << k)][k]);
    ret = f(ret, tbl[i + len - (1 << k)][j + len - (1 <<
        k)][k]);
    return ret;
  }
}
int main() {
  for(int i = 0; i < n; i++)
    for(int j = 0; j < m; j++)
      cin >> st2 :: tbl[i][j][0];
  st2 :: init(n, m);
  cout << st2 :: query(x, y, s); // x, y, x + s - 1, y +
      s - 1
}
```

3 Number Theory

3.1 Big MOD

```cpp
LL bigmod(LL x, LL n, LL mod) {
  if(n == -1) n = mod - 2;
  LL ans = 1;
  while(n) {
    if((n & 1)) ans = (ans * x) % mod;
    n >>= 1;
    x = (x * x) % mod;
  }
  return ans;
}
```

3.2 Bitwise Sieve

```cpp
const int nmax = 1e8 + 1;
int mark[(nmax >> 6) + 1];
vector<int> primes;
#define isSet(n, pos) (bool)((n) & (1LL << (pos)))
#define Set(n, pos) ((n) | (1LL << (pos)))
void sieve(int n) {
  for (int i = 3; i * i <= n; i += 2) {
    if (isSet(mark[i >> 6], (i >> 1) & 31) == 0) {
      for (int j = i * i; j <= n; j += (i << 1)) mark[j
          >> 6] = Set(mark[j >> 6], (j >> 1) & 31);
    }
  }
  primes.push_back(2);
  for (int i = 3; i <= n; i += 2) {
    if (isSet(mark[i >> 6], (i >> 1) & 31) == 0) primes.
        push_back(i);
  }
}
```

3.3 Chinese Reminder Theorem

```cpp
using LL = long long;
using PLL = pair<LL, LL>;
// given a, b will find solutions for, ax + by = 1
tuple<LL, LL, LL> EGCD(LL a, LL b) {
  if (b == 0)
    return {1, 0, a};
  else {
    auto [x, y, g] = EGCD(b, a % b);
    return {y, x - a / b * y, g};
  }
}
// given modulo equations, will apply CRT
PLL CRT(vector<PLL> &v) {
  LL V = 0, M = 1;
  for (auto &[v, m] : v) { // value % mod
    auto [x, y, g] = EGCD(M, m);
    if ((v - V) % g != 0) return {-1, 0};
    V += x * (v - V) / g % (m / g) * M, M *= m / g;
    V = (V % M + M) % M;
  }
  return make_pair(V, M);
}
```

3.4 Combinatorics

```cpp
/* Given n boxes, each box has cnt[i] different (
    distinct) items,
   you can take only 1 object from each box. how many
       different combinations
   of choices are there */
LL caLL(LL box, LL take, vector<LL> &cnt) {
  vector<vector<int>> DP(box + 1, vector<int>(take + 2));
  dp[0][0] = 1, dp[0][1] = cnt[0];
  for (int s = 0; s <= take; s++) {
    for (int idx = 0; idx < box; idx++) {
      dp[idx + 1][s] = (dp[idx + 1][s] + dp[idx][s]);
      dp[idx + 1][s + 1] = (dp[idx + 1][s + 1] + dp[idx][
          s] * cnt[idx + 1][s]);
    }
  }
  return dp[box - 1][take];
}
```

3.5 Divisor

```cpp
// calculate divisor in range[1,n]
LL sum_in_range(LL n) {
  return n * (n + 1) / 2;
}
LL sum_all_divisors(LL n) {
  LL ans = 0;
  for(LL i=1;i*i<=n;i++) {
    LL hello = i * (n / i - i + 1);
    LL world = sum_in_range(n / i) - sum_in_range(i);
    ans += hello + world;
  }
  return ans;
}
```

3.6 Eulars Totient Function

```cpp
int phi(int n) {
  int ret = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % p == 0) n /= i;
      ret -= ret / i;
    }
  }
  if (n > 1) ret -= ret / n;
  return ret;
}
void phi_in_range() {
  int N = 1e6, phi[N + 1];
  for (int i = 0; i <= N; i++) phi[i] = i;
```

```
  for (int i = 2; i <= N; i++) {
    if (phi[i] != i) continue;
    for (int j = i; j <= N; j += i) {
      phi[j] -= phi[j] / i;
    }
  }
}

#some important properties of phi
phi(p) = p-1 ,where p is a prime number
phi(a*b) = phi(a)*phi(b) ,where a and b are co-prime
phi(a*b) = phi(a)*phi(b)*(gcd(a,b)/phi(gcd(a,b))) ,for
    any number
phi(p^k) = p^k - p^(k-1) ,where p is a prime number, '^'
    indicates power
Sum of values of totient functions of all divisors of n
    is equal to n.
```

3.7 FFT

```
using CD = complex <double>;
typedef long long LL;
const double PI = acos(-1.0L);

int N;
vector<int> perm;
vector<CD> wp[2];
void precalculate(int n) {
  assert((n & (n - 1)) == 0), N = n;
  perm = vector<int>(N, 0);
  for (int k = 1; k < N; k <<= 1) {
    for (int i = 0; i < k; i++) {
      perm[i] <<= 1;
      perm[i + k] = 1 + perm[i];
    }
  }
  wp[0] = wp[1] = vector<CD>(N);
  for (int i = 0; i < N; i++) {
    wp[0][i] = CD(cos(2 * PI * i / N), sin(2 * PI * i / N
        ));
    wp[1][i] = CD(cos(2 * PI * i / N), -sin(2 * PI * i /
        N));
  }
}
void fft(vector<CD> &v, bool invert = false) {
  if (v.size() != perm.size()) precalculate(v.size());
  for (int i = 0; i < N; i++)
    if (i < perm[i]) swap(v[i], v[perm[i]]);
  for (int len = 2; len <= N; len *= 2) {
    for (int i = 0, d = N / len; i < N; i += len) {
      for (int j = 0, idx = 0; j < len / 2; j++, idx += d
          ) {
        CD x = v[i + j];
        CD y = wp[invert][idx] * v[i + j + len / 2];
        v[i + j] = x + y;
        v[i + j + len / 2] = x - y;
      }
    }
  }
  if (invert) {
    for (int i = 0; i < N; i++) v[i] /= N;
  }
}
void pairfft(vector<CD> &a, vector<CD> &b, bool invert =
    false) {
  int N = a.size();
  vector<CD> p(N);
  for (int i = 0; i < N; i++) p[i] = a[i] + b[i] * CD(0,
      1);
  fft(p, invert);
  p.push_back(p[0]);
  for (int i = 0; i < N; i++) {
    if (invert) {
      a[i] = CD(p[i].real(), 0);
      b[i] = CD(p[i].imag(), 0);
    } else {
      a[i] = (p[i] + conj(p[N - i])) * CD(0.5, 0);
      b[i] = (p[i] - conj(p[N - i])) * CD(0, -0.5);
    }
  }
}
vector<LL> multiply(const vector<LL> &a, const vector<LL>
     &b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> fa(a.begin(), a.end()), fb(b.begin(), b.end
      ());
  fa.resize(n);
  fb.resize(n);
  //       fft(fa); fft(fb);
  pairfft(fa, fb);
  for (int i = 0; i < n; i++) fa[i] = fa[i] * fb[i];
  fft(fa, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) ans[i] = round(fa[i].real()
      );
  return ans;
}
const int M = 1e9 + 7, B = sqrt(M) + 1;
vector<LL> anyMod(const vector<LL> &a, const vector<LL> &
    b) {
  int n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  vector<CD> al(n), ar(n), bl(n), br(n);
  for (int i = 0; i < a.size(); i++) al[i] = a[i] % M / B
      , ar[i] = a[i] % M % B;
  for (int i = 0; i < b.size(); i++) bl[i] = b[i] % M / B
      , br[i] = b[i] % M % B;
  pairfft(al, ar);
  pairfft(bl, br);
  //       fft(al); fft(ar); fft(bl); fft(br);
  for (int i = 0; i < n; i++) {
    CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
    CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
    al[i] = ll;
    ar[i] = lr;
    bl[i] = rl;
    br[i] = rr;
  }
  pairfft(al, ar, true);
  pairfft(bl, br, true);
  //       fft(al, true); fft(ar, true); fft(bl, true);
      fft(br, true);
  vector<LL> ans(n);
  for (int i = 0; i < n; i++) {
    LL right = round(br[i].real()), left = round(al[i].
        real());
    ;
    LL mid = round(round(bl[i].real()) + round(ar[i].real
        ()));
    ans[i] = ((left % M) * B * B + (mid % M) * B + right)
         % M;
  }
  return ans;
}
```

3.8 LargePrime

```
vector<int> sieve(const int N, const int Q = 17, const
    int L = 1 << 15) {
```

```
static const int rs[] = {1, 7, 11, 13, 17, 19, 23, 29};
struct P {
  P(int p) : p(p) {}
  int p; int pos[8];
};
auto approx_prime_count = [] (const int N) -> int {
  return N > 60184 ? N / (log(N) - 1.1)
                   : max(1., N / (log(N) - 1.11)) + 1;
};

const int v = sqrt(N), vv = sqrt(v);
vector<bool> isp(v + 1, true);
for (int i = 2; i <= vv; ++i) if (isp[i]) {
  for (int j = i * i; j <= v; j += i) isp[j] = false;
}

const int rsize = approx_prime_count(N + 30);
vector<int> primes = {2, 3, 5}; int psize = 3;
primes.resize(rsize);

vector<P> sprimes; size_t pbeg = 0;
int prod = 1;
for (int p = 7; p <= v; ++p) {
  if (!isp[p]) continue;
  if (p <= Q) prod *= p, ++pbeg, primes[psize++] = p;
  auto pp = P(p);
  for (int t = 0; t < 8; ++t) {
    int j = (p <= Q) ? p : p * p;
    while (j % 30 != rs[t]) j += p << 1;
    pp.pos[t] = j / 30;
  }
  sprimes.push_back(pp);
}

vector<unsigned char> pre(prod, 0xFF);
for (size_t pi = 0; pi < pbeg; ++pi) {
  auto pp = sprimes[pi]; const int p = pp.p;
  for (int t = 0; t < 8; ++t) {
    const unsigned char m = ~(1 << t);
    for (int i = pp.pos[t]; i < prod; i += p) pre[i] &=
        m;
  }
}

const int block_size = (L + prod - 1) / prod * prod;
vector<unsigned char> block(block_size); unsigned char*
     pblock = block.data();
const int M = (N + 29) / 30;

for (int beg = 0; beg < M; beg += block_size, pblock -=
     block_size) {
  int end = min(M, beg + block_size);
  for (int i = beg; i < end; i += prod) {
    copy(pre.begin(), pre.end(), pblock + i);
  }
  if (beg == 0) pblock[0] &= 0xFE;
  for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
    auto& pp = sprimes[pi];
    const int p = pp.p;
    for (int t = 0; t < 8; ++t) {
      int i = pp.pos[t]; const unsigned char m = ~(1 <<
          t);
      for (; i < end; i += p) pblock[i] &= m;
      pp.pos[t] = i;
    }
  }
  for (int i = beg; i < end; ++i) {
    for (int m = pblock[i]; m > 0; m &= m - 1) {
      primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
    }
  }
}
assert(psize <= rsize);
while (psize > 0 && primes[psize - 1] > N) --psize;
primes.resize(psize);
return primes;
}
```

3.9 Matrix

```
int n;
struct Matrix{
  vector<vector<LL>> Mat = vector<vector<LL>>(n, vector<
      LL>(n));
  // memset(Mat,0,sizeof(Mat));
  Matrix operator*(const Matrix &other){
   Matrix product;
   for (int i = 0; i < n; i++){
     for (int j = 0; j < n; j++){
       for (int k = 0; k < n; k++){
         LL temp = ((Mat[i][k] % mod)*(other.Mat[k][j]%
             mod))%mod;
         product.Mat[i][j] = (product.Mat[i][j] % mod +
             temp % mod) % mod;
       }
     }
   }
   return product;
  }
};
Matrix MatExpo(Matrix a, int p){
  Matrix product;
  for (int i = 0; i < n; i++)
    product.Mat[i][i] = 1;
  while (p > 0){
    if (p % 2) product = product * a;
    p /= 2;
    a = a * a;
  }
  return product;
}
```

3.10 Mint

```
int mint::M = 1e9 + 7;
class mint {
 private:
  int value;
  static int M;

  void normalize() {
    value %= M;
    if (value < 0) value += M;
  }

  int mpow(int x, int n) const {
    if (n == -1) n = M - 2;
    int ans = 1;
    while (n) {
      if (n & 1) ans = (ans * x) % M;
      n >>= 1;
      x = (x * x) % M;
    }
    return ans;
  }

 public:
  mint() : value(0){};
  mint(int value) : value(value) { normalize(); }
```

```cpp
  mint& operator=(int value) {
    this->value = value;
    normalize();
    return *this;
  }

  mint operator+(const mint& other) const { return mint(
      value + other.value); }
  mint operator+(int other) const { return mint(value +
      other); }
  mint operator-(const mint& other) const { return mint(
      value - other.value); }
  mint operator-(int other) const { return mint(value -
      other); }
  mint operator*(const mint& other) const { return mint(
      value * other.value); }
  mint operator*(int other) const { return mint(value *
      other); }
  mint operator/(const mint& other) const { return *this
      * mpow(other.value, -1); }
  mint operator/(int other) const { return *this * mpow(
      other, -1); }

  mint& operator+=(const mint& other) {
    value += other.value;
    normalize();
    return *this;
  }
  mint& operator+=(int other) {
    value += other;
    normalize();
    return *this;
  }
  mint& operator-=(const mint& other) {
    value -= other.value;
    normalize();
    return *this;
  }
  mint& operator-=(int other) {
    value -= other;
    normalize();
    return *this;
  }
  mint& operator*=(const mint& other) {
    value *= other.value;
    normalize();
    return *this;
  }
  mint& operator*=(int other) {
    value *= other;
    normalize();
    return *this;
  }
  mint& operator/=(const mint& other) {
    value = (value * mpow(other.value, -1));
    normalize();
    return *this;
  }
  mint& operator/=(int other) {
    value = (value * mpow(other, -1));
    normalize();
    return *this;
  }

  mint pow(int expo) const { return mint(mpow(value, expo
      )); }
  mint pow(const mint& expo) const { return mint(mpow(
      value, expo.value)); }

  friend ostream& operator<<(ostream& os, const mint& var
      ) {
    os << var.value;
    return os;
  }
  friend istream& operator>>(istream& is, mint& var) {
    is >> var.value;
    var.normalize();
    return is;
  }

  int get() { return value; }
};

namespace com {
mint fact[N], inv[N], inv_fact[N];
void init() {
  fact[0] = inv_fact[0] = 1;
  for (int i = 1; i < N; i++) {
    inv[i] = i == 1 ? 1 : inv[i - mod % i] * (mod / i +
        1);
    fact[i] = fact[i - 1] * i;
    inv_fact[i] = inv_fact[i - 1] * inv[i];
  }
}

mint C(int n, int r) { return (r < 0 or r > n) ? mint(0)
    : fact[n] * inv_fact[r] * inv_fact[n - r]; }
mint P(int n, int r) { return (r < 0 or r > n) ? mint(0)
    : fact[n] * inv_fact[n - r]; }

mint C(mint& n, int r) { return C(n.get(), r); }
mint P(mint& n, int r) { return P(n.get(), r); }

mint C(int& n, mint& r) { return C(n, r.get()); }
mint P(int& n, mint& r) { return P(n, r.get()); }

mint C(mint& n, mint& r) { return C(n.get(), r.get()); }
mint P(mint& n, mint& r) { return P(n.get(), r.get()); }
} // namespace com
using namespace com;
```

3.11 NOD and SOD

```cpp
// NUMBER = p_1^a_1 * p_2^a_2 .... p_n^a_n
LL NOD = 1, SOD = 1, POD = 1, POWER = 1;
for(int i = 0; i < n; i++) {
  LL p, a; cin >> p >> a;
  NOD = (NOD * (a + 1)) % MOD;
  SOD = ((SOD * (bigmod(p, a + 1, MOD) + MOD - 1)) % MOD
      * inv[p - 1]) % MOD;
  POD = bigmod(POD, a + 1, MOD) * bigmod(bigmod(x, a * (a
       + 1) / 2, MOD), POWER, MOD) % MOD;
  POWER = (POWER * (a + 1)) % (MOD - 1);
}
cout << NOD << ' ' << SOD << ' ' << POD << '\n';
// FULL TEMPLATE
using LL = long long;
using ULL = unsigned long long;
namespace sieve {
const int N = 1e7;
vector<int> primes;
int spf[N + 5], phi[N + 5], NOD[N + 5], cnt[N + 5], POW[N
     + 5];
bool prime[N + 5];
int SOD[N + 5];
void init() {
  fill(prime + 2, prime + N + 1, 1);
  SOD[1] = NOD[1] = phi[1] = spf[1] = 1;
```

```
  for (LL i = 2; i <= N; i++) {
    if (prime[i]) {
      primes.push_back(i), spf[i] = i;
      phi[i] = i - 1;
      NOD[i] = 2, cnt[i] = 1;
      SOD[i] = i + 1, POW[i] = i;
    }
    for (auto p : primes) {
      if (p * i > N or p > spf[i]) break;
      prime[p * i] = false, spf[p * i] = p;
      if (i % p == 0) {
        phi[p * i] = p * phi[i];
        NOD[p * i] = NOD[i] / (cnt[i] + 1) * (cnt[i] + 2)
            ,
                cnt[p * i] = cnt[i] + 1;
        SOD[p * i] = SOD[i] / SOD[POW[i]] * (SOD[POW[i]]
            + p * POW[i]),
                POW[p * i] = p * POW[i];
        break;
      } else {
        phi[p * i] = phi[p] * phi[i];
        NOD[p * i] = NOD[p] * NOD[i], cnt[p * i] = 1;
        SOD[p * i] = SOD[p] * SOD[i], POW[p * i] = p;
      }
    }
  }
}

// CSOD
LL csod(LL n) {
  LL ans = 0;
  for(LL i = 2; i * i <= n; ++i) {
    LL j = n / i;
    ans += (i + j) * (j - i + 1) / 2;
    ans += i * (j - i);
  }
  return ans;
}
summation of NOD(d)[d|n] = product of g(e_k + 1)[n=p_k^
    a_k]
g(x) = x * (x + 1) / 2
```

3.12 Pollard rho

```
namespace rho{
  inline LL mul(LL a, LL b, LL mod) {
    LL result = 0;
    while (b) {
      if (b & 1) result = (result + a) % mod;
      a = (a + a) % mod;
      b >>= 1;
    }
    return result;
  }
  inline LL bigmod(LL num,LL pow,LL mod){
    LL ans = 1;
    for( ; pow > 0; pow >>= 1, num = mul(num, num, mod))
      if(pow&1) ans = mul(ans,num,mod);
    return ans;
  }
  inline bool is_prime(LL n){
    if(n < 2 or n % 6 % 4 != 1) return (n|1) == 3;
    LL a[] = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022};
    LL s = __builtin_ctzll(n-1), d = n >> s;
    for(LL x: a){
      LL p = bigmod(x % n, d, n), i = s;
      for( ; p != 1 and p != n-1 and x % n and i--; p =
          mul(p, p, n));
      if(p != n-1 and i != s) return false;
    }
    return true;
  }
  LL f(LL x, LL n) {
    return mul(x, x, n) + 1;
  }
  LL get_factor(LL n) {
    LL x = 0, y = 0, t = 0, prod = 2, i = 2, q;
    for( ; t++ %40 or __gcd(prod, n) == 1; x = f(x, n), y
        = f(f(y, n), n) ){
      (x == y) ? x = i++, y = f(x, n) : 0;
      prod = (q = mul(prod, max(x,y) - min(x,y), n)) ? q
          : prod;
    }
    return __gcd(prod, n);
  }
  void _factor(LL n, map <LL, int> &res) {
    if(n == 1) return;
    if(is_prime(n)) res[n]++;
    else {
      LL x = get_factor(n);
      _factor(x, res);
      _factor(n / x, res);
    }
  }
  map <LL, int> factorize(LL n){
    map <LL, int> res;
    if(n < 2)  return res;
    LL small_primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23,
        29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
        79, 83, 89, 97 };
    for (LL p: small_primes)
      for( ; n % p == 0; n /= p, res[p]++);
    _factor(n, res);
    return res;
  }
}
```

3.13 Sieve

```
const int N = 10000000;
vector <int> lp(N), pr;
for (int i = 2; i < N; i++) {
  if (lp[i] == 0) {
    lp[i] = i;
    pr.push_back (i);
  }
  for (int j = 0; i * pr[j] < N; j++) {
    lp[i * pr[j]] = pr[j];
    if (pr[j] == lp[i]) break;
  }
}
```

3.14 nCr

```
namespace com {
LL fact[N], inv[N], inv_fact[N];
void init() {
  fact[0] = inv_fact[0] = 1;
  for (int i = 1; i < N; i++) {
    inv[i] = i == 1 ? 1 : (LL)inv[i - mod % i] * (mod / i
        + 1) % mod;
    fact[i] = (LL)fact[i - 1] * i % mod;
    inv_fact[i] = (LL)inv_fact[i - 1] * inv[i] % mod;
  }
}
LL C(int n, int r) { return (r < 0 or r > n) ? 0 : fact[n
    ] * inv_fact[r] % mod * inv_fact[n - r] % mod; }
}  // namespace com
```

3.15 ntt

```
const LL N = 1 << 18;
const LL MOD = 786433;

vector<LL> P[N];
LL rev[N], w[N | 1], a[N], b[N], inv_n, g;
LL Pow(LL b, LL p) {
  LL ret = 1;
  while (p) {
    if (p & 1) ret = (ret * b) % MOD;
    b = (b * b) % MOD;
    p >>= 1;
  }
  return ret;
}
LL primitive_root(LL p) {
  vector<LL> factor;
  LL phi = p - 1, n = phi;
  for (LL i = 2; i * i <= n; i++) {
    if (n % i) continue;
    factor.emplace_back(i);
    while (n % i == 0) n /= i;
  }
  if (n > 1) factor.emplace_back(n);
  for (LL res = 2; res <= p; res++) {
    bool ok = true;
    for (LL i = 0; i < factor.size() && ok; i++)
      ok &= Pow(res, phi / factor[i]) != 1;
    if (ok) return res;
  }
  return -1;
}
void prepare(LL n) {
  LL sz = abs(31 - __builtin_clz(n));
  LL r = Pow(g, (MOD - 1) / n);
  inv_n = Pow(n, MOD - 2);
  w[0] = w[n] = 1;
  for (LL i = 1; i < n; i++) w[i] = (w[i - 1] * r) % MOD;
  for (LL i = 1; i < n; i++)
    rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
}
void NTT(LL *a, LL n, LL dir = 0) {
  for (LL i = 1; i < n - 1; i++)
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (LL m = 2; m <= n; m <<= 1) {
    for (LL i = 0; i < n; i += m) {
      for (LL j = 0; j < (m >> 1); j++) {
        LL &u = a[i + j], &v = a[i + j + (m >> 1)];
        LL t = v * w[dir ? n - n / m * j : n / m * j] %
            MOD;
        v = u - t < 0 ? u - t + MOD : u - t;
        u = u + t >= MOD ? u + t - MOD : u + t;
      }
    }
  }
  if (dir)
    for (LL i = 0; i < n; i++) a[i] = (inv_n * a[i]) %
        MOD;
}
vector<LL> mul(vector<LL> p, vector<LL> q) {
  LL n = p.size(), m = q.size();
  LL t = n + m - 1, sz = 1;
  while (sz < t) sz <<= 1;
  prepare(sz);

  for (LL i = 0; i < n; i++) a[i] = p[i];
  for (LL i = 0; i < m; i++) b[i] = q[i];
  for (LL i = n; i < sz; i++) a[i] = 0;
  for (LL i = m; i < sz; i++) b[i] = 0;

  NTT(a, sz);
  NTT(b, sz);
  for (LL i = 0; i < sz; i++) a[i] = (a[i] * b[i]) % MOD;
  NTT(a, sz, 1);

  vector<LL> c(a, a + sz);
  while (c.size() && c.back() == 0) c.pop_back();
  return c;
}
```

4 Graph

4.1 Bellman Ford

```
void bellmanford(int n, int m, vector<int> edge[], int
    dist[], int src){
  fill(dist, dist + n, INT_MAX);
  dist[src] = 0;
  int i, j, k;
  vector<int> v;
  for (i = 0; i < n; i++){
    for (j = 0; j < m; j++) {
      v = edge[j];
      if (dist[v[1]] > dist[v[0]] + v[2])
        dist[v[1]] = dist[v[0]] + v[2];
    }
  }
  for (j = 0; j < m; j++){ // For checking negative loop
    v = edge[j];
    if (dist[v[1]] > dist[v[0]] + v[2]){
      fill(dist, dist + n, INT_MIN); // Negative loop
          detected
      return;
    }
  }
}
```

4.2 BridgeTree

```
vector<PLL> g[N];
vector<int> ng[N];
int disc[N], low[N], mark[N], vis[N], timer = 1;
;

void find_bridge(int u, int p) {
  disc[u] = low[u] = timer++;
  bool fl = 1;
  for (auto [v, id] : g[u]) {
    if (v == p && fl) {
      fl = 0;
      continue;
    }
    if (disc[v]) {
      low[u] = min(low[u], disc[v]);
    } else {
      find_bridge(v, u);
      low[u] = min(low[u], low[v]);
      if (disc[u] < low[v]) {
        mark[id] = 1;
      }
    }
  }
}

void colorComponents(int u, int color) {
  if (vis[u]) return;
  vis[u] = color;
  for (auto [v, id] : g[u]) {
    if (mark[id]) continue;
    colorComponents(v, color);
  }
```

```
}

void solve() {
  int n, m;
  cin >> n >> m;
  vector<PLL> edges;
  for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    edges.push_back({u, v});
    g[u].push_back({v, i});
    g[v].push_back({u, i});
  }
  find_bridge(1, 0);
  int color = 1;
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) colorComponents(i, color++);
  }
  for (int i = 0; i < m; i++) {
    if (mark[i]) {
      ng[vis[edges[i].first]].push_back(vis[edges[i].
          second]);
      ng[vis[edges[i].second]].push_back(vis[edges[i].
          first]);
    }
  }
}
```

4.3 DSU, MST

```
class DSU {
  vector<int> parent, size;

 public:
  DSU(int n) : parent(n + 1), size(n + 1, 1) { iota(
      parent.begin(), parent.end(), 0); }
  int root(int u) {
    if (parent[u] == u) return u;
    return parent[u] = root(parent[u]);
  }
  bool same(int u, int v) { return root(u) == root(v); }
  void merge(int u, int v) {
    u = root(u), v = root(v);
    if (u == v) return;
    if (size[u] < size[v]) swap(u, v);
    parent[v] = u, size[u] += size[v];
  }
};

int kruskal(vector<tuple<int, int, int>> edges, int n) {
  sort(edges.begin(), edges.end());
  DSU mst(n);
  int cost = 0;
  for (auto &[w, u, v] : edges) {
    if (mst.same(u, v)) continue;
    mst.merge(u, v);
    cost += w;
  }
  return cost;
}
// PRIM'S SPANNING TREE (MST)
DIJKSTRA code...
start from a node, and push nodes which are not marked
    popped edges weight are taken
```

4.4 Dijkstra

```
struct node {
  int to;
  LL weight;
  bool operator<(const node &a) const {
    return weight > a.weight;
  }
};
vector<node> adj[N];
void dijkstra(int src, vector<LL> &dist, vector<int> &
    parent) {
  parent.assign(n + 1, -1);
  priority_queue<node> pq;
  pq.push({src, 0});
  dist[src] = 0;
  parent[src] = -1;
  while(!pq.empty()) {
    auto cur = pq.top(); pq.pop();
    for(auto next : adj[cur.to]) {
      if(dist[next.to] > dist[cur.to] + next.weight) {
        dist[next.to] = dist[cur.to] + next.weight;
        pq.push({next.to, dist[next.to]});
        parent[next.to] = cur.to;
      }
    }
  }
}
```

4.5 ETT, VT

```
struct euler_tour {
  int time = 0;
  tree &T;
  int n;
  vector<int> start, finish, level, par;
  euler_tour(tree &T, int root = 0)
      : T(T), n(T.n), start(n), finish(n), level(n), par(
          n) {
    time = 0;
    call(root);
  }
  void call(int node, int p = -1) {
    if (p != -1) level[node] = level[p] + 1;
    start[node] = time++;
    for (int e : T[node])
      if (e != p) call(e, node);
    par[node] = p;
    finish[node] = time++;
  }
  bool isAncestor(int node, int par) {
    return start[par] <= start[node] and finish[par] >=
        finish[node];
  }
  int subtreeSize(int node) { return finish[node] - start
      [node] + 1 >> 1; }
};
tree virtual_tree(vector<int> &nodes, lca_table &table,
    euler_tour &tour) {
  sort(nodes.begin(), nodes.end(),
      [&](int x, int y) { return tour.start[x] < tour.
          start[y]; });
  int n = nodes.size();
  for (int i = 0; i + 1 < n; i++)
    nodes.push_back(table.lca(nodes[i], nodes[i + 1]));
  sort(nodes.begin(), nodes.end());
  nodes.erase(unique(nodes.begin(), nodes.end()), nodes.
      end());
  sort(nodes.begin(), nodes.end(),
      [&](int x, int y) { return tour.start[x] < tour.
          start[y]; });
  n = nodes.size();
  stack<int> st;
  st.push(0);
  tree ans(n);
  for (int i = 1; i < n; i++) {
```

```cpp
    while (!tour.isAncestor(nodes[i], nodes[st.top()]))
      st.pop();
    ans.addEdge(st.top(), i);
    st.push(i);
  }
  return ans;
}
set<int> getCenters(tree &T) {
  int n = T.n;
  vector<int> deg(n), q;
  set<int> s;
  for (int i = 0; i < n; i++) {
    deg[i] = T[i].size();
    if (deg[i] == 1) q.push_back(i);
    s.insert(i);
  }
  for (vector<int> t; s.size() > 2; q = t) {
    for (auto x : q) {
      for (auto e : T[x])
        if (--deg[e] == 1) t.push_back(e);
      s.erase(x);
    }
  }
  return s;
}
```

4.6 HLD

```cpp
int sub[nmax], par[nmax], depth[nmax];
vector<int> adj[nmax];

void dfs_sz(int u, int p) {
  sub[u] = 1, par[u] = p;
  depth[u] = (p == -1) ? 0 : depth[p] + 1;
  int mx = 0; /// HLD
  for (auto &v : adj[u]) {
    if (v == p) continue;
    dfs_sz(v, u);
    sub[u] += sub[v];
    if (sub[v] > mx) mx = sub[v], swap(v, adj[u][0]); ///
        HLD
  }
}

int head[nmax];
int st[nmax], en[nmax], clk;
int dfsarr[nmax]; /// segtree will be built on this

void dfs_hld(int u, int p) {
  st[u] = ++clk;
  /// put stuff in dfarr here
  dfsarr[clk] = val[u]; /// node specific value

  head[u] = (p != -1 && adj[p][0] == u) ? head[p] : u; //
      / HLD
  for (auto &v : adj[u]) {
    if (v == p) continue;
    dfs_hld(v, u);
  }
  en[u] = clk;
}

int lca(int a, int b) {
  for (; head[a] != head[b]; b = par[head[b]])
    if (depth[head[a]] > depth[head[b]]) swap(a, b);
  if (depth[a] > depth[b]) swap(a, b);
  return a;
}

// process node u upto it's ancestor a
// if excl is true, a will not be processed
int chainProcess(int a, int u, bool excl = false) {
  for (; head[u] != head[a]; u = par[head[u]]) {
    func(st[head[u]], st[u]); // processing
                              // query(1, 1, n, st[head[u
                                  ]], st[u])
  }
  func(st[a] + excl, st[u]); // processing
}

// process path from node u to node v, if order matters
    will be tough
// if excl is true lca will not be processed
int pathProcess(int a, int b, bool excl) {
  for (; head[a] != head[b]; b = par[head[b]]) {
    if (depth[head[a]] > depth[head[b]]) swap(a, b);
    func(st[head[b]], st[b]);
  }
  if (depth[a] > depth[b]) swap(a, b);
  func(st[a] + excl, st[b]);
}
```

4.7 K th shortest path

```cpp
void K_shortest(int n, int m) {
  int st, des, k, u, v;
  LL w;
  scanf("%d%d%d", &st, &des, &k);
  st--, des--;
  vector<vector<pii> > edges(n);
  for (int i = 0; i < m; i++) {
    scanf("%d%d%lld", &u, &v, &w);
    u--, v--;
    edges[u].push_back({w, v});
  }
  vector<vector<LL> > dis(n, vector<LL>(k + 1, 1e8));
  vector<int> vis(n);
  priority_queue<pii, vector<pii>, greater<pii> > q;

  q.emplace(0LL, st);
  while (!q.empty()) {
    v = q.top().second, w = q.top().first;
    q.pop();
    if (vis[v] >= k) continue;
    // for the varient, check if this path is greater
        than previous, if not, continue
    // if(vis[v]>0 && w == dis[v][vis[v]-1]) continue;
    dis[v][vis[v]] = w;
    vis[v]++;
    for (auto nd : edges[v]) {
      q.emplace(w + nd.first, nd.second);
    }
  }
  LL ans = dis[des][k - 1];
  if (ans == 1e8) ans = -1;
  printf("%lld\n", ans);
}
```

4.8 LCA, CD

```cpp
struct Tree {
  vector<vector<int>> adj;
  Tree(int N) : adj(N + 1) {}
  void addEdges(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
};

class LCA {
  int N, K;
```

```
  vector<vector<int>> &adj, anc;
  vector<int> level;

 public:
  LCA(Tree &tree) : adj(tree.adj) {
    N = tree.adj.size() - 1;
    K = 33 - __builtin_clz(N);
    anc.assign(N + 1, vector<int>(K));
    level.assign(N + 1, 0);
    initLCA(1);
  }

  void initLCA(int u, int p = 0) {
    anc[u][0] = p;
    level[u] = level[p] + 1;
    for (int i = 1; i < K; i++) {
      anc[u][i] = anc[anc[u][i - 1]][i - 1];
    }
    for (auto v : adj[u])
      if (v != p) {
        initLCA(v, u);
      }
  }
  int getAnc(int u, int k) {
    for (int i = K - 1; i >= 0; i--)
      if (k & (1 << i)) u = anc[u][i];
    return u;
  }
  int lca(int u, int v) {
    if (level[u] > level[v]) swap(u, v);
    v = getAnc(v, level[v] - level[u]);

    if (u == v) return u;
    for (int i = K - 1; i >= 0; i--) {
      if (anc[u][i] != anc[v][i]) u = anc[u][i], v = anc[
          v][i];
    }
    return anc[u][0];
  }
  int dis(int u, int v) { return level[u] + level[v] - 2
       * level[lca(u, v)]; }
};

class CD {
  vector<vector<int>> adj;
  vector<int> sub;
  vector<bool> blocked;
  int N;

 public:
  CD(Tree &tree) : adj(tree.adj) {
    N = tree.adj.size() - 1;
    blocked.assign(N + 1, 0);
    sub.assign(N + 1, 0);
    compute();
  }
  void compute(int u = 1, int p = 0) {
    sub[u] = 1;
    for (auto v : adj[u])
      if (v != p) {
        compute(v, u);
        sub[u] += sub[v];
      }
  }
  int centroid(int u, int p = 0) {
    int tot = sub[u];
    for (auto v : adj[u]) {
      if (v == p || blocked[v]) continue;
      if (2 * sub[v] > tot) {
        sub[u] = tot - sub[v];
        sub[v] = tot;
        return centroid(v, u);
      }
    }
    return u;
  }

  int count(int u, int p) { // calculate ans
  }
  void update(int u, int p) { // update
  }

  int decompose(int u = 1) {
    u = centroid(u);
    blocked[u] = 1;
    int ans = 0;

    /////  Do something here  //// count() update()
    for (auto v : adj[u])
      if (!blocked[v]) {
        ans += count(v, u);
        update(v, u);
      }
    /// reset updates here

    for (auto v : adj[u])
      if (!blocked[v]) {
        decompose(v);
      }
    return ans;
  }
};
```

4.9 block cut tree

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

using namespace std;

const int N = 200010;

bitset <N> art, good;
vector <int> g[N], tree[N], st, comp[N];
int n, m, ptr, cur, in[N], low[N], id[N];

void dfs (int u, int from = -1) {
  in[u] = low[u] = ++ptr;
  st.emplace_back(u);
  for (int v : g[u]) if (v ^ from) {
    if (!in[v]) {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= in[u]) {
        art[u] = in[u] > 1 or in[v] > 2;
        comp[++cur].emplace_back(u);
        while (comp[cur].back() ^ v) {
          comp[cur].emplace_back(st.back());
          st.pop_back();
        }
      }
    } else {
      low[u] = min(low[u], in[v]);
    }
  }
}

void buildTree() {
  ptr = 0;
  for (int i = 1; i <= n; ++i) {
```

```
    if (art[i]) id[i] = ++ptr;
  }
  for (int i = 1; i <= cur; ++i) {
    int x = ++ptr;
    for (int u : comp[i]) {
      if (art[u]) {
        tree[x].emplace_back(id[u]);
        tree[id[u]].emplace_back(x);
      } else {
        id[u] = x;
      }
    }
  }
}

int main() {
  cin >> n >> m;
  while (m--) {
    int u, v;
    scanf("%d %d", &u, &v);
    g[u].emplace_back(v);
    g[v].emplace_back(u);
  }
  for (int i = 1; i <= n; ++i) {
    if (!in[i]) dfs(i);
  }
  buildTree();
  for (int i = 1; i <= ptr; ++i) {
    cout << i << " --> ";
    for (int j : tree[i]) cout << j << " "; cout << '\n';
  }
  return 0;
}
```

4.10 strongly connected component

```
bool vis[N];
vector<int> adj[N], adjr[N];
vector<int> order, component;
// tp = 0, finding topo order,
// tp = 1, reverse edge traversal
void dfs(int u, int tp = 0) {
  vis[u] = true;
  if (tp) component.push_back(u);
  auto& ad = (tp ? adjr : adj);
  for (int v : ad[u])
    if (!vis[v]) dfs(v, tp);
  if (!tp) order.push_back(u);
}
int main() {
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) dfs(i);
  }
  memset(vis, 0, sizeof vis);
  reverse(order.begin(), order.end());
  for (int i : order) {
    if (!vis[i]) {
      // one component is found
      dfs(i, 1), component.clear();
    }
  }
}
```

4.11 tree isomorphism

```
// has to include bigmod
LL Hash(int u, int p) {
  vector<LL> childrenHash;
  for (auto v : adj[u]) if (v != p)
    childrenHash.add(Hash(v, u));
  sort(all(childrenHash));
  LL nodeHash = 0;
  for (int i = 0; i < childrenHash.size(); i++)
    nodeHash = (nodeHash + childrenHash[i] * bigmod(SEED,
        i, MOD)) % MOD;
  return nodeHash;
}
```

5 String

5.1 KMP

```
int KMP(vector<int> &a, vector<int> &b) { // number of
    occurance of a in b
  vector<int> pi(n);
  for (int i = 1, j = 0; i < n; i++) {
    while (j && a[i] != a[j]) j = pi[j - 1];
    if (a[i] == a[j]) j++;
    pi[i] = j;
  }
  int ans = 0;
  for (int i = 0, j = 0; i < m; i++) {
    while (j && b[i] != a[j]) j = pi[j - 1];
    if (a[j] == b[i]) j++;
    if (j == n) ans++, j = pi[j - 1];
  }
  return ans;
}
```

5.2 Manacher

```
void Manacher() {
  vector<int> d1(n);
  // d[i] = number of palindromes taking s[i] as center
  for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
    while (0 <= i - k && i + k < n && s[i - k] == s[i + k
        ]) k++;
    d1[i] = k--;
    if (i + k > r) l = i - k, r = i + k;
  }

  vector<int> d2(n);
  // d[i] = number of palindromes taking s[i-1] and s[i]
     as center
  for (int i = 0, l = 0, r = -1; i < n; i++) {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i +
        1);
    while (0 <= i - k - 1 && i + k < n && s[i - k - 1] ==
         s[i + k]) k++;
    d2[i] = k--;
    if (i + k > r) l = i - k - 1, r = i + k;
  }
}
```

5.3 SuffixArray

```
void inducedSort (const vector <int> &vec, int val_range,
     vector <int> &SA, const vector <int> &sl, const
    vector <int> &lms_idx) {
  vector <int> l(val_range, 0), r(val_range, 0);
  for (int c : vec) {
    ++r[c]; if (c + 1 < val_range) ++l[c + 1];
  }
  partial_sum(l.begin(), l.end(), l.begin());
  partial_sum(r.begin(), r.end(), r.begin());
  fill(SA.begin(), SA.end(), -1);
  for (int i = lms_idx.size() - 1; i >= 0; --i) SA[--r[
      vec[lms_idx[i]]]] = lms_idx[i];
  for (int i : SA) if (i > 0 and sl[i - 1]) SA[l[vec[i -
      1]]++] = i - 1;
  fill(r.begin(), r.end(), 0);
  for (int c : vec) ++r[c];
  partial_sum(r.begin(), r.end(), r.begin());
```

```
  for (int k = SA.size() - 1, i = SA[k]; k; --k, i = SA[k
      ]) {
    if (i and !sl[i - 1]) SA[--r[vec[i - 1]]] = i - 1;
  }
}

vector <int> suffixArray (const vector <int> &vec, int
    val_range) {
  const int n = vec.size();
  vector <int> sl(n), SA(n), lms_idx;
  for (int i = n - 2; i >= 0; --i) {
    sl[i] = vec[i] > vec[i + 1] or (vec[i] == vec[i + 1]
        and sl[i + 1]);
    if (sl[i] and !sl[i + 1]) lms_idx.emplace_back(i + 1)
        ;
  }
  reverse(lms_idx.begin(), lms_idx.end());
  inducedSort(vec, val_range, SA, sl, lms_idx);
  vector <int> new_lms_idx(lms_idx.size()), lms_vec(
      lms_idx.size());
  for (int i = 0, k = 0; i < n; ++i) {
    if (SA[i] > 0 and !sl[SA[i]] and sl[SA[i] - 1])
        new_lms_idx[k++] = SA[i];
  }
  int cur = 0; SA[n - 1] = 0;
  for (int k = 1; k < new_lms_idx.size(); ++k) {
    int i = new_lms_idx[k - 1], j = new_lms_idx[k];
    if (vec[i] ^ vec[j]) {
      SA[j] = ++cur; continue;
    }
    bool flag = 0;
    for (int a = i + 1, b = j + 1; ; ++a, ++b) {
      if (vec[a] ^ vec[b]) {
        flag = 1; break;
      }
      if ((!sl[a] and sl[a - 1]) or (!sl[b] and sl[b -
          1])) {
        flag = !(!sl[a] and sl[a - 1] and !sl[b] and sl[b
            - 1]); break;
      }
    }
    SA[j] = flag ? ++cur : cur;
  }
  for (int i = 0; i < lms_idx.size(); ++i) lms_vec[i] =
      SA[lms_idx[i]];
  if (cur + 1 < lms_idx.size()) {
    auto lms_SA = suffixArray(lms_vec, cur + 1);
    for (int i = 0; i < lms_idx.size(); ++i) new_lms_idx[
        i] = lms_idx[lms_SA[i]];
  }
  inducedSort(vec, val_range, SA, sl, new_lms_idx);
      return SA;
}

vector <int> getSuffixArray (const string &s, const int
    LIM = 128) {
  vector <int> vec(s.size() + 1);
  copy(begin(s), end(s), begin(vec)); vec.back() = '#';
  auto ret = suffixArray(vec, LIM);
  ret.erase(ret.begin()); return ret;
}

// build RMQ on it to get LCP of any two suffix
vector <int> getLCParray (const string &s, const vector <
    int> &SA) {
  int n = s.size(), k = 0;
  vector <int> lcp(n), rank(n);
  for (int i = 0; i < n; ++i) rank[SA[i]] = i;
  for (int i = 0; i < n; ++i, k ? --k : 0) {
    if (rank[i] == n - 1) {
      k = 0; continue;
    }
    int j = SA[rank[i] + 1];
    while (i + k < n and j + k < n and s[i + k] == s[j +
        k]) ++k;
    lcp[rank[i]] = k;
  }
  lcp[n - 1] = 0; return lcp;
}
```

5.4 Z

```
vector<int> z(string const& s) {
    int n = size(s);
    vector<int> z(n);
    int x = 0, y = 0;
    for (int i = 1; i < n; i++) {
     z[i] = max(0, min(z[i - x], y - i + 1));
     while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
      x = i, y = i + z[i], z[i]++;
     }
    }
    return z;
}
```

5.5 double hashing

```
/*
    **********************************************************

Simple Library for String Hashing, Uses Double Hash.
Hash(abc........z) = a*p^n + b*p^(n-1) + ...... + z

In order to convert to Single Hash -
    o Delete operator overloads and fix reduce()
    o Replace all PLL with LL
    o Change mp pairs to appropriate value
Or set M2 = 1, which should be nearly as fast.

Some Primes:
1000000007, 1000000009, 1000000861, 1000099999 ( < 2^30 )
1088888881, 1111211111, 1500000001, 1481481481 ( < 2^31 )
2147483647 (2^31-1),

Author: anachor
\**********************************************************
    */
typedef pair<LL, LL> PLL;
namespace Hashing {
    #define ff first
    #define ss second
    const PLL M = {1e9+7, 1e9+9};    ///Should be large
        primes
    const LL base = 1259;            ///Should be larger
        than alphabet size
    const int N = 1e6+7;             ///Highest length of
        string
    PLL operator+ (const PLL& a, LL x) {return {a.ff + x,
         a.ss + x};}
    PLL operator- (const PLL& a, LL x) {return {a.ff - x,
         a.ss - x};}
    PLL operator* (const PLL& a, LL x) {return {a.ff * x,
         a.ss * x};}
    PLL operator+ (const PLL& a, PLL x) {return {a.ff + x
        .ff, a.ss + x.ss};}
    PLL operator- (const PLL& a, PLL x) {return {a.ff - x
        .ff, a.ss - x.ss};}
    PLL operator* (const PLL& a, PLL x) {return {a.ff * x
        .ff, a.ss * x.ss};}
```

```cpp
PLL operator% (const PLL& a, PLL m) {return {a.ff % m
    .ff, a.ss % m.ss};;}
ostream& operator<<(ostream& os, PLL hash) {
    return os<<"("<<hash.ff<<", "<<hash.ss<<")";
}
PLL pb[N];    ///powers of base mod M
///Call pre before everything
void hashPre() {
    pb[0] = {1,1};
    for (int i=1; i<N; i++)  pb[i] = (pb[i-1] * base)
        %M;
}
///Calculates hashes of all prefixes of s including
    empty prefix
vector<PLL> hashList(string s) {
    int n = s.size();
    vector<PLL> ans(n+1);
    ans[0] = {0,0};
    for (int i=1; i<=n; i++) ans[i] = (ans[i-1] *
        base + s[i-1])%M;
    return ans;
}
///Calculates hash of substring s[l..r] (1 indexed)
PLL substringHash(const vector<PLL> &hashlist, int l,
    int r) {
    return (hashlist[r]+(M-hashlist[l-1])*pb[r-l+1])%
        M;
}
///Calculates Hash of a string
PLL Hash (string s) {
    PLL ans = {0,0};
    for (int i=0; i<s.size(); i++) ans=(ans*base + s[
        i])%M;
    return ans;
}
///Tested on https://toph.co/p/palindromist
///appends c to string
PLL append(PLL cur, char c) {
    return (cur*base + c)%M;
}
///Tested on https://toph.co/p/palindromist
///prepends c to string with size k
PLL prepend(PLL cur, int k, char c) {
    return (pb[k]*c + cur)%M;
}
///Tested on https://toph.co/p/chikongunia
///replaces the i-th (0-indexed) character from right
    from a to b;
PLL replace(PLL cur, int i, char a, char b) {
    return cur + pb[i] * (M+b-a)%M;
}
///Erases c from front of the string with size len
PLL pop_front(PLL hash, int len, char c) {
    return (hash + pb[len-1]*(M-c))%M;
}
///Tested on https://toph.co/p/palindromist
///concatenates two strings where length of the right
    is k
PLL concat(PLL left, PLL right, int k) {
    return (left*pb[k] + right)%M;
}
PLL power (const PLL& a, LL p) {
    if (p==0)  return {1,1};
    PLL ans = power(a, p/2);
    ans = (ans * ans)%M;
    if (p%2)   ans = (ans*a)%M;
    return ans;
}
PLL inverse(PLL a) {
    if (M.ss == 1) return power(a, M.ff-2);
    return power(a, (M.ff-1)*(M.ss-1)-1);
}
///Erases c from the back of the string
PLL invb = inverse({base, base});
PLL pop_back(PLL hash, char c) {
    return ((hash-c+M)*invb)%M;
}
///Tested on https://toph.co/p/palindromist
///Calculates hash of string with size len repeated
    cnt times
///This is O(log n). For O(1), pre-calculate inverses
PLL repeat(PLL hash, int len, LL cnt) {
    PLL mul = ((pb[len*cnt]-1+M) * inverse(pb[len]-1+
        M))%M;
    PLL ans = (hash*mul);
    if (pb[len].ff == 1)  ans.ff = hash.ff*cnt;
    if (pb[len].ss == 1)  ans.ss = hash.ss*cnt;
    return ans%M;
}
}
/// Solves https://judge.yosupo.jp/problem/
    enumerate_palindromes
using namespace Hashing;
vector<PLL> forwardHash, backwardHash;
int n;
bool check(int l, int r) {
    return substringHash(forwardHash, l, r) ==
        substringHash(backwardHash, n+1-r, n+1-l);
}
```

6 Divide and Conquer

6.1 maximum subarray sum

```cpp
array<LL, 3> maxSubArraySum(std::vector<LL> &v, LL n) {
  LL max_so_far = -INF, max_ending_here = 0, start = 0,
      end = 0, s = 0;
  for(int i = 0; i < n; i++) {
    max_ending_here += v[i];
    if(max_so_far < max_ending_here) {
      max_so_far = max_ending_here;
      start = s, end = i;
    }
    if(max_ending_here < 0) {
      max_ending_here = 0;
      s = i + 1;
    }
  }
  return {max_so_far, start, end};
}
```

7 DP

7.1 CatalanDp

```cpp
const int nmax = 1e4 + 1;
const int mod = 1000000007;
int catalan[nmax + 1];
// comb formula: ((2n)Cn)-((2n)C(n-1)) = (1/(n+1))*((2n)
    Cn)
void genCatalan(int n) {
  catalan[0] = catalan[1] = 1;
  for (int i = 2; i <= n; i++) {
    catalan[i] = 0;
    for (int j = 0; j < i; j++) {
      catalan[i] += (catalan[j] * catalan[i - j - 1]) %
          mod;
      if (catalan[i] >= mod) {
        catalan[i] -= mod;
      }
    }
```

```
 }
}
```

7.2 Coin Change

```
void coin(){ //given different types of coin how many way
     number x can be formed?
    int n,x,mod=1e9+7; cin>>n>>x;
    int a[n], dp[x+1]={};
    for(int i=0;i<n;i++){
        cin>>a[i];
        if(a[i]<=x) dp[a[i]]=1;
    }
    for(int i=1;i<=x;i++){
        for(int j=0;j<n;j++){
            if(i>= a[j]){
                dp[i] += dp[i-a[j]];
                dp[i] %= mod;
            }
        }
    }
    cout<<dp[x]<<ln;
}
```

7.3 DearrangementDP

```
const int nmax = 2e5 + 1;
int drng[nmax + 1];

void gen_drng(int n) {
 drng[2] = 1;
 for (int i = 3; i <= n; i++) {
  drng[i] = ((i - 1ll) * ((drng[i - 2] + drng[i - 1]) %
      mod)) % mod;
 }
}
```

7.4 Knapsack

```
/*
for 1 ---->
    1<=N<=100
    1<=W<=105
    1<=wi<=W
    1<=vi<=1e9
for 2 ---->
    1<=N<=100
    1<=W<=1e9
    1<=wi<=W
    1<=vi<=1e3
*/
int n,W,v[101],w[101],dp[101][N];

int ks(int i,int W){
    if(i>=n) return 0;
    if(dp[i][W]!=-1) return dp[i][W];
    if(W<w[i]) return dp[i][W]=ks(i+1,W);
    else return dp[i][W]=max(ks(i+1,W),ks(i+1,W-w[i])+v[i
        ]);
}
void solve(){
    cin>>n>>W;
    for(int i=0;i<n;i++){
        cin>>w[i]>>v[i];
    }
    cout<<ks(0,W)<<ln;
}

int knpsk2(int i,int val){
    if(val==0) return 0;
    if(i>=n) return INT_MAX;
    if(dp[i][val]!=-1) return dp[i][val];
    int b = ks(i+1,val);
    if(val-v[i]>=0) b = min(b,ks(i+1,val-v[i]) + w[i]);
    return dp[i][val] = b;
}
void solve(){
    cin>>n>>W;
    for(int i=0;i<n;i++){
        cin>>w[i]>>v[i];
    }
    int ans=0, sm=accumulate(v,v+n,0LL);;
    for(int j=sm;j>=0;j--){
        if(ks(0,j)<=W){
            cout<<j<<ln;
            break;
        }
    }
}
```

7.5 SOS

```
/*
Given a fixed array A of 2^N integers, we need to
    calculate for all x function F(x) = Sum of all A[i]
    such that x&i = i, i.e., i is a subset of x.
*/
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
 dp[mask][-1] = A[mask]; //handle base case separately (
      leaf states)
 for(int i = 0;i < N; ++i){
  if(mask & (1<<i))
   dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
  else
   dp[mask][i] = dp[mask][i-1];
 }
 F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
 F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N)
    ; ++mask){
 if(mask & (1<<i))
  F[mask] += F[mask^(1<<i)];
}
```

7.6 grundy

```
/* single pile game-> greedy or game dp multiple pile
     game and disjunctive(before playing, choose 1 pile)
     -> NIM game
else-> Grundy(converts n any game piles to n NIM piles)
----------------------------------------------------------

grundy(x)->the smallest nonreachable grundy value
----------------------------------------------------------

there are n pile of games and k type of moves.
if XOR(grundy(games)) == 0: losing state else winning
     state */
vector<int> moves, dp;
int mex(vector<int> &a) {
set<int> b(a.begin(), a.end());
for (int i = 0; ; ++i)
 if (!b.count(i)) return i;
}
int grundy(int x) {
 if (dp[x] != -1) return dp[x];
 vector<int> reachable;
 for (auto m : moves) {
  if (x - m < 0) continue;
```

```
  int val = grundy(x - m);
  reachable.push_back(val);
 }
 return dp[x] = mex(reachable);
}
```

8 Geometry

8.1 2D everything

```
using LL = long long;
using ULL = unsigned long long;

const double PI = acos(-1), EPS = 1e-10;
template <typename DT> DT sq(DT x) {return x * x; }
template <typename DT> int dcmp(DT x) { return fabs(x) <
    EPS ? 0 : (x<0 ? -1 : 1);}

template <typename DT>
class point{
 public:
 DT x,y;
 point() = default;
 point(DT x, DT y): x(x), y(y) {};
 template <typename X> point(point <X> p): x(p.x), y(p.y
     ) {};
 point operator + (const point &rhs) const { return
     point(x + rhs.x, y + rhs.y); }
 point operator - (const point &rhs) const { return
     point(x - rhs.x, y - rhs.y); }
 point operator * (const point &rhs) const { return
     point(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x)
     ;}
 point operator / (const point &rhs) const { return *
     this * point(rhs.x, - rhs.y) / ~(rhs);}

 point operator * (DT M)   const { return point(M * x, M
      * y); }
 point operator / (DT M)   const { return point(x / M, y
      / M); }
 bool operator < (point rhs) const { return x < rhs.x
      or (x == rhs.x and y < rhs.y); }
 bool operator == (const point &rhs) const { return x ==
      rhs.x and y == rhs.y; }
 bool operator <= (const point &rhs) const { return *
      this < rhs or *this == rhs; }
 bool operator != (const point &rhs) const { return x !=
      rhs.x or y != rhs.y; }
 DT operator & (const point &rhs) const { return x *
     rhs.y - y * rhs.x; } // cross product
 DT operator ^ (const point &rhs) const { return x *
     rhs.x + y * rhs.y; } // dot product
 DT operator ~()   const {return sq(x) + sq(y); }
                    //square of norm
 point operator - () const {return *this * -1; }
 friend istream& operator >> (istream &is, point &p) {
     return is >> p.x >> p.y; }
 friend ostream& operator << (ostream &os, const point &
     p) { return os << p.x << " " << p.y; }

 friend DT DisSq(const point &a, const point &b){ return
      sq(a.x-b.x) + sq(a.y-b.y); }
 friend DT TriArea(const point &a, const point &b, const
      point &c) { return (b-a) & (c-a); }
 friend DT UTriArea(const point &a, const point &b,
     const point &c) { return abs(TriArea(a, b, c)); }
 friend bool Collinear(const point &a, const point &b,
     const point &c) { return UTriArea(a, b, c) < EPS; }
 friend double Angle(const point &u) { return atan2(u.y,
      u.x); }
 friend double Angle(const point &a, const point &b) {
   double ans = Angle(b) - Angle(a);
   return ans <= -PI ? ans + 2*PI : (ans > PI ? ans - 2*
       PI : ans);
 }
 point Perp(const point &a){ return point(-a.y, a.x); }
 point Conj(const point &a){ return point(a.x, -a.y); }
};
template <typename DT> using polygon = vector <point <DT
    >>;
template <typename DT>
class polarComp {
 point <DT> O, dir;
 bool half(point <DT> p) {
   return dcmp(dir & p) < 0 || (dcmp(dir & p) == 0 &&
       dcmp(dir ^ p) > 0);
 }
 public:
 polarComp(point <DT> O = point(0, 0), point <DT> dir =
     point(1, 0)) : O(O), dir(dir) {}
 bool operator() (point <DT> p, point <DT> q) {
   return make_tuple(half(p), 0) < make_tuple(half(q), (
       p & q));
 }
}; // given a pivot point and an initial direction, sorts
    by Angle with the given direction

template <typename DT>
class line{
 public:
 point <DT> dir, O; // direction of vector and starting
     point
 line(point <DT> p,point <DT> q): dir(q - p), O(p) {};

 bool Contains(const point <double> &p){
   return fabs(p - O & dir ) < EPS;
 } // checks whether the line Contains a certain point
 template <typename XT> point <XT> At(XT t){
   return point <XT> (dir) * t + O;
 } // inserts value of t in the vector representation,
     finds the point which is O + Dir*t
 double AtInv(const point <double> &p){
   return abs(dir.x) > 0 ? (p - O).x / dir.x : (p - O).y
       / dir.y;
 } // if the line Contains a point, gives the value t
     such that, p = O+Dir*t
 line Perp(point <DT> p){
   return line(p, p + (-dir.y,dir.x));
 }
 point <DT> ProjOfPoint(const point <DT> &P) {
   return O + dir * ((P - O) ^ dir) / (~dir);
 }
 double DisOfPoint(const point <DT> &P) {
   return fabs(dir & (P - O))/sqrt(~(dir));
 }
 friend bool Parallel(line& L, line& R){
   return fabs(R.dir & L.dir) < EPS;
 }
 friend int Intersects(line& L, line& R){
   return Parallel(L, R) ? R.Contains(L.O) ? -1 : 0 : 1;
 }
 friend pair <double, double> IntersectionAt(line &L,
     line &R){
   double r = double((L.O - R.O) & L.dir)/(R.dir & L.dir
       );
   double l = double((R.O - L.O) & R.dir)/(L.dir & R.dir
       );
   return {l, r};
 }
```

```
  friend pair <int, point<double>> IntersectionPoint(line
      L, line R,int _L = 0, int _R = 0){
    // _L and _R can be 0 to 3, 0 is a normal line, 3 is
       a segment, 1 and 2 are rays (considered bitwise)
    int ok = Intersects(L, R);
    if(ok == 0) return {0, {0, 0}};
    if(ok == 1){
      auto [l,r] = IntersectionAt(L, R);
      if(l < (0-EPS) and _L & 2 ) return {0, {0, 0}};
      if(l > (1+EPS) and _L & 1) return {0, {0, 0}};
      if(r < (0-EPS) and _R & 2 ) return {0, {0, 0}};
      if(r > (1+EPS) and _R & 1) return {0, {0, 0}};
      return {1, L.At(l)};
    }
    return {-1, {0,0}}; // they are the same line
  }
};
template <typename DT>
class circle {
  public:
  point <DT> O; DT R;
  circle(const point <DT> &O = {0, 0}, DT R = 0) : O(O),
      R(R) {}
  // the next two make sense only on circle <double>
  circle(const point <DT> &A, const point <DT> &B, const
      point <DT> &C){
  point <DT> X = (A + B) / 2, Y = (B + C) / 2, d1 = Perp(
      A - B), d2 = Perp(B - C);
    O = IntersectionPoint(line(X, d1), line(Y, d2)).
       second;
    R = sqrt(~(O - A));
  }
  circle(const point <DT> &A, const point <DT> &B, DT R){
    point <DT> X = (A + B) / 2, d = Perp(A - B);
      d = d * (R / sqrt(~(d)));
      O = X + d;
      R = sqrt(~(O - A));
    }
    double SectorArea(double ang) {
      // Area of a sector of cicle
      return ang* R * R * .5;
    }
    double SectorArea(const point <DT> &a, const point <
       DT> &b) {
      return SectorArea(Angle(a - O, b - O));
    }
    double ChordArea(const point <DT> &a, const point <DT
       > &b) {
      // Area between sector and its chord
      return SectorArea(a, b) - 0.5 * TriArea(O, a, b);
    }
    int Contains(const point <DT> &p){
      // 0 for outside, 1 for inside, -1 for on the
         circle
      DT d = DisSq(O, p);
      return d > R * R ? 0 : (d == R * R ? -1 : 1);
    }
    friend tuple <int, point <DT>, point <DT>>
       IntersectionPoint(const circle &a,const circle &b)
       {
      if(a.R == b.R and a.O == b.O) return {-1, {0, 0}, {0,
         0}};
      double d = sqrt(DisSq(a.O, b.O));
      if(d > a.R + b.R or d < fabs(a.R - b.R)) return {0,
         {0, 0}, {0, 0}};
      double z = (sq(a.R) + sq(d) - sq(b.R)) / (2 * d);
      double y = sqrt(sq(a.R) - sq(z));
      point <DT> O = b.O - a.O, h = Perp(O) * (y / sqrt(~O)
         );
      O = a.O + O * (z / sqrt(~O));
      return make_tuple(1 + (~(h) > EPS), O - h, O + h);
    }
    friend tuple <int, point <DT>, point <DT>>
       IntersectionPoint(const circle &C, line <DT> L) {
      point <DT> P = L.ProjOfPoint(C.O);
      double D = DisSq(C.O, P);
      if(D > C.R * C.R) return {0, {0, 0}, {0, 0}};
      double x = sqrt(C.R * C.R - D);
      point <DT> h = L.dir * (x / sqrt(~L.dir));
      return {1 + (x > EPS), P - h, P + h};
    }
    double SegmentedArea(point <DT> &a, point <DT> &b) {
      // signed area of the intersection between the circle
         and triangle OAB
      double ans = SectorArea(a, b);
      line <DT> L(a, b);
      auto [cnt, p1, p2] = IntersectionPoint(*this, L);
      if(cnt < 2) return ans;
      double t1 = L.AtInv(p1), t2 = L.AtInv(p2);
      if(t2 < 0 or t1 > 1) return ans;
      if(t1 < 0) p1 = a;
      if(t2 > 1) p2 = b;
      return ans - ChordArea(p1, p2);
    }
};
namespace polygon_algo{
  template <typename DT> polygon <DT> ConvexHull(polygon
      <DT> &PT){
    sort(PT.begin(), PT.end());
    int m = 0, n = PT.size();
    polygon <DT> hull(n + n + 2);
    for(int i = 0; i < n; i++){
      for( ; m > 1 and TriArea(hull[m-2], hull[m-1], PT[i
         ]) <= 0; m-- );
      hull[m++] = PT[i];
    }
    for(int i = n - 2, k = m; i >= 0; i--){
      for( ; m > k and TriArea(hull[m - 2], hull[m - 1],
         PT[i]) <= 0; m--);
      hull[m++] = PT[i];
    }
    if(n > 1) m--;
    while(hull.size() > m) hull.pop_back();
    return hull;
  }
  template <typename DT> double MinimumBoundingBox(
      polygon <DT> P){
    auto p = ConvexHull(P);
    int n = p.size();
    double area = 1e20 + 5;
    for(int i = 0, l = 1, r = 1, u = 1 ; i < n ; i++){
      point <DT> edge = (p[(i+1)%n]- p[i])/sqrt(DisSq(p[i
         ], p[(i+1)%n]));
      for( ; (edge ^ p[r%n]-p[i]) < (edge ^ p[(r+1)%n] -
         p[i]); r++);
      for( ; u<r || (edge & p[u%n] - p[i]) < (edge & p[(u
         +1)%n] - p[i]); u++) ;
      for( ; l<u || (edge ^ p[l%n] - p[i]) > (edge ^ p[(l
         +1)%n] - p[i]); l++);
      double w = (edge ^ p[r%n]-p[i]) - (edge ^ p[l%n] -
         p[i]);
      double h = UTriArea(p[u%n], p[i], p[(i+1)%n])/sqrt(
         DisSq(p[i], p[(i+1)%n]));
      area = min(area,w*h);
    }
    if(area>1e19) area = 0;
    return area;
  }
```

```cpp
  template <typename DT> DT FarthestPairOfPoints(polygon
      <DT> p){
    p = ConvexHull(p);
    int n = p.size();
    DT ans = -1e9;
    for(int i = 0, j = 1; i < n; i++) {
      for( ; UTriArea(p[i], p[(i + 1) % n], p[(j + 1) % n
          ]) > UTriArea(p[i], p[(i + 1) % n], p[j]) ; j =
           (j + 1) % n ) ;
      ans = max(ans, DisSq(p[i], p[j]));
      ans = max(ans, DisSq(p[(i + 1) % n], p[j]));
    }
    return ans; // will return square of the answer.
  }
  template <typename DT> int PointInConvexPolygon(polygon
       <int> :: iterator b, polygon <int> :: iterator e,
       const point <DT> &O){
    polygon <int> :: iterator lo = b + 2, hi = e - 1, ans
         = e;
    while(lo <= hi) {
      auto mid = lo + (hi - lo) / 2;
      if(TriArea(*b, O, *mid) >= 0) ans = mid, hi = mid -
           1;
      else lo = mid + 1;
    }
    if (ans == e or abs(UTriArea(*b, *(ans - 1), *ans) -
        UTriArea(*b, *(ans - 1), O) - UTriArea(*b, *ans,
        O) - UTriArea(*(ans - 1), *ans, O)) > EPS) return
         0;
    else return (Collinear(*b, *(b + 1), O) or Collinear
        (*(e - 1), *b, O) or Collinear(*(ans), *(ans - 1)
        , O)) ? -1 : 1;
  } // 0 for outside, -1 for on border, 1 for inside
  template <typename DT> int PointInPolygon(polygon <DT>
      &P, point <DT> pt) {
    int n = P.size();
    int cnt = 0;
    for(int i = 0, j = 1; i < n; i++, j = (j + 1) % n) {
      if(TriArea(pt, P[i], P[j]) == 0 and min(P[i], P[j])
           <= pt and pt <= max(P[i], P[j])) return -1;
      cnt += ((P[j].y >= pt.y) - (P[i].y >= pt.y)) *
          TriArea(pt, P[i], P[j]) > 0;
    }
    return cnt & 1;
  }
}
using namespace polygon_algo;

// CLOSEST PAIR OF POINTS
template <typename DT> Dis(point <DT> a, point <DT> b){
    return ~(a - b);
}
template <typename DT>
DT Closest_Distance(vector <point <DT>> &v) {
  int n = v.size();
  sort(v.begin(), v.end());
  auto cmp = [](point <DT> a, point <DT> b) {return (a.y
       < b.y || (a.y == b.y && a.x < b.x));};
  set <point <DT>, decltype(cmp)> s(cmp);
  DT best = 1e18;
  int j = 0;
  for (int i = 0; i < n; i++) {
    while (sq(v[i].x - v[j].x) >= best) {
      s.erase(v[j]);
      j = (j + 1) % n;
    }
    DT d = best;
    auto it1 = s.lower_bound( point <DT>(v[i].x, v[i].y -
         d) );
    auto it2 = s.upper_bound( point <DT>(v[i].x, v[i].y +
         d) );
    for (auto it = it1; it != it2; it++) best = min(best,
         Dis(v[i], *it));
    s.insert(v[i]);
  }
  return best;
}
```

8.2 3D

```cpp
template <typename DT>
class Point {
public:
  DT x, y, z;
  Point(){};
  Point(DT x, DT y, DT z) : x(x), y(y), z(z) {}
  template <typename X> Point(Point<X> p) : x(p.x), y(p.y
      ), z(p.z) {}
  Point operator + (const Point &rhs) const { return
      Point(x + rhs.x, y + rhs.y, z + rhs.z); }
  Point operator - (const Point &rhs) const { return
      Point(x - rhs.x, y - rhs.y, z - rhs.z); }
  Point operator * (DT M) const { return Point(M * x, M *
       y, M * z); }
  Point operator / (DT M) const { return Point(x / M, y /
       M, z / M); }
  // cross product
  Point operator & (const Point &rhs) const { return
      Point(y * rhs.z - z * rhs.y,z * rhs.x - x * rhs.z,x
       * rhs.y - y * rhs.x); }
  // dot product
  DT operator ^ (const Point &rhs) const { return x * rhs
      .x + y * rhs.y + z * rhs.z; }
  bool operator == (const Point &rhs) const { return x ==
       rhs.x && y == rhs.y && z == rhs.z; }
  bool operator != (const Point &rhs) const { return !(*
      this == rhs); }
  friend std::istream& operator >> (std::istream &is,
      Point &p) { return is >> p.x >> p.y >> p.z; }
  friend std::ostream& operator << (std::ostream &os,
      const Point &p) { return os << p.x << " " << p.y <<
       " " << p.z; }
  friend DT DisSq(const Point &a, const Point &b) {
      return (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y -
       b.y) + (a.z - b.z)*(a.z - b.z); }
};

optional < Point <double> > ray_intersects_triangle(const
     Point<double> &origin,const Point<double> &
     ray_vector,const array <Point<double>, 3> &triangle)
      {
  constexpr double epsilon = std::numeric_limits<double
      >::epsilon();
  auto [A, B, C] = triangle;
  Point<double> edge1 = B - A;
  Point<double> edge2 = C - A;
  Point<double> ray_cross_e2 = ray_vector & edge2;
  double det = edge1 ^ ray_cross_e2;
  if (det > -epsilon && det < epsilon) return {}; // Ray
      is parallel to this triangle.
  double inv_det = 1.0 / det;
  Point<double> s = ray_origin - A;
  double u = inv_det * (s ^ ray_cross_e2);
  if (u < 0 || u > 1) return {};
  Point<double> s_cross_e1 = s & edge1;
  double v = inv_det * (ray_vector ^ s_cross_e1);
  if (v < 0 || u + v > 1) return {};
  // Compute t to find the intersection Point
```

```
  double t = inv_det * (edge2 ^ s_cross_e1);
  if (t > epsilon) return ray_origin + ray_vector * t; //
       ray intersection
  else return {}; // Line intersection but not ray
      intersection
}
// HOW TO IMPLEMENT
// auto tmp = ray_intersects_triangle (origin, ray, v[i])
    ;
// if (tmp.has_value ()) Point <double>
    intersection_point = tmp.value ();
```

8.3 convex

```
/// minkowski sum of two polygons in O(n)
Polygon minkowskiSum(Polygon A, Polygon B) {
  int n = A.size(), m = B.size();
  rotate(A.begin(), min_element(A.begin(), A.end()), A.
      end());
  rotate(B.begin(), min_element(B.begin(), B.end()), B.
      end());

  A.push_back(A[0]);
  B.push_back(B[0]);
  for (int i = 0; i < n; i++) A[i] = A[i + 1] - A[i];
  for (int i = 0; i < m; i++) B[i] = B[i + 1] - B[i];

  Polygon C(n + m + 1);
  C[0] = A.back() + B.back();
  merge(A.begin(), A.end() - 1, B.begin(), B.end() - 1, C
      .begin() + 1,
       polarComp(Point(0, 0), Point(0, -1)));
  for (int i = 1; i < C.size(); i++) C[i] = C[i] + C[i -
      1];
  C.pop_back();
  return C;
}
// finds the rectangle with minimum area enclosing a
    convex polygon and
// the rectangle with minimum perimeter enclosing a
    convex polygon
// Tf Ti Same
pair<Tf, Tf> rotatingCalipersBoundingBox(const Polygon &p
    ) {
  using Linear::distancePointLine;
  int n = p.size();
  int l = 1, r = 1, j = 1;
  Tf area = 1e100;
  Tf perimeter = 1e100;
  for (int i = 0; i < n; i++) {
    Point v = (p[(i + 1) % n] - p[i]) / length(p[(i + 1)
        % n] - p[i]);
    while (dcmp(dot(v, p[r % n] - p[i]) - dot(v, p[(r +
        1) % n] - p[i])) < 0)
      r++;
    while (j < r || dcmp(cross(v, p[j % n] - p[i]) -
                         cross(v, p[(j + 1) % n] - p[i])) <
                              0)
      j++;
    while (l < j ||
           dcmp(dot(v, p[l % n] - p[i]) - dot(v, p[(l + 1)
                % n] - p[i])) > 0)
      l++;
    Tf w = dot(v, p[r % n] - p[i]) - dot(v, p[l % n] - p[
        i]);
    Tf h = distancePointLine(p[j % n], Line(p[i], p[(i +
        1) % n]));
    area = min(area, w * h);
    perimeter = min(perimeter, 2 * w + 2 * h);
  }
  return make_pair(area, perimeter);
}
// returns the left side of polygon u after cutting it by
    ray a->b
Polygon cutPolygon(Polygon u, Point a, Point b) {
  using Linear::lineLineIntersection;
  using Linear::onSegment;

  Polygon ret;
  int n = u.size();
  for (int i = 0; i < n; i++) {
    Point c = u[i], d = u[(i + 1) % n];
    if (dcmp(cross(b - a, c - a)) >= 0) ret.push_back(c);
    if (dcmp(cross(b - a, d - c)) != 0) {
      Point t;
      lineLineIntersection(a, b - a, c, d - c, t);
      if (onSegment(t, Segment(c, d))) ret.push_back(t);
    }
  }
  return ret;
}
// returns true if point p is in or on triangle abc
bool pointInTriangle(Point a, Point b, Point c, Point p)
    {
  return dcmp(cross(b - a, p - a)) >= 0 && dcmp(cross(c -
       b, p - b)) >= 0 &&
        dcmp(cross(a - c, p - c)) >= 0;
}
// pt must be in ccw order with no three collinear points
// returns inside = -1, on = 0, outside = 1
int pointInConvexPolygon(const Polygon &pt, Point p) {
  int n = pt.size();
  assert(n >= 3);

  int lo = 1, hi = n - 1;
  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;
    if (dcmp(cross(pt[mid] - pt[0], p - pt[0])) > 0)
      lo = mid;
    else
      hi = mid;
  }

  bool in = pointInTriangle(pt[0], pt[lo], pt[hi], p);
  if (!in) return 1;

  if (dcmp(cross(pt[lo] - pt[lo - 1], p - pt[lo - 1])) ==
       0) return 0;
  if (dcmp(cross(pt[hi] - pt[lo], p - pt[lo])) == 0)
       return 0;
  if (dcmp(cross(pt[hi] - pt[(hi + 1) % n], p - pt[(hi +
      1) % n])) == 0)
    return 0;
  return -1;
}
// Extreme Point for a direction is the farthest point in
     that direction
// u is the direction for extremeness
int extremePoint(const Polygon &poly, Point u) {
  int n = (int)poly.size();
  int a = 0, b = n;
  while (b - a > 1) {
    int c = (a + b) / 2;
    if (dcmp(dot(poly[c] - poly[(c + 1) % n], u)) >= 0 &&
        dcmp(dot(poly[c] - poly[(c - 1 + n) % n], u)) >=
             0) {
      return c;
    }
```

```cpp
    bool a_up = dcmp(dot(poly[(a + 1) % n] - poly[a], u)) >= 0;
    bool c_up = dcmp(dot(poly[(c + 1) % n] - poly[c], u)) >= 0;
    bool a_above_c = dcmp(dot(poly[a] - poly[c], u)) > 0;

    if (a_up && !c_up)
      b = c;
    else if (!a_up && c_up)
      a = c;
    else if (a_up && c_up) {
      if (a_above_c)
        b = c;
      else
        a = c;
    } else {
      if (!a_above_c)
        b = c;
      else
        a = c;
    }
  }

  if (dcmp(dot(poly[a] - poly[(a + 1) % n], u)) > 0 &&
      dcmp(dot(poly[a] - poly[(a - 1 + n) % n], u)) > 0)
    return a;
  return b % n;
}
// For a convex polygon p and a line l, returns a list of segments
// of p that touch or intersect line l.
// the i'th segment is considered (p[i], p[(i + 1) modulo |p|])
// #1 If a segment is collinear with the line, only that is returned
// #2 Else if l goes through i'th point, the i'th segment is added
// Complexity: O(lg |p|)
vector<int> lineConvexPolyIntersection(const Polygon &p, Line l) {
  assert((int)p.size() >= 3);
  assert(l.a != l.b);

  int n = p.size();
  vector<int> ret;

  Point v = l.b - l.a;
  int lf = extremePoint(p, rotate90(v));
  int rt = extremePoint(p, rotate90(v) * Ti(-1));
  int olf = orient(l.a, l.b, p[lf]);
  int ort = orient(l.a, l.b, p[rt]);

  if (!olf || !ort) {
    int idx = (!olf ? lf : rt);
    if (orient(l.a, l.b, p[(idx - 1 + n) % n]) == 0)
      ret.push_back((idx - 1 + n) % n);
    else
      ret.push_back(idx);
    return ret;
  }
  if (olf == ort) return ret;

  for (int i = 0; i < 2; ++i) {
    int lo = i ? rt : lf;
    int hi = i ? lf : rt;
    int olo = i ? ort : olf;

    while (true) {
      int gap = (hi - lo + n) % n;
      if (gap < 2) break;

      int mid = (lo + gap / 2) % n;
      int omid = orient(l.a, l.b, p[mid]);
      if (!omid) {
        lo = mid;
        break;
      }
      if (omid == olo)
        lo = mid;
      else
        hi = mid;
    }
    ret.push_back(lo);
  }
  return ret;
}
// Calculate [ACW, CW] tangent pair from an external point
constexpr int CW = -1, ACW = 1;
bool isGood(Point u, Point v, Point Q, int dir) {
  return orient(Q, u, v) != -dir;
}
Point better(Point u, Point v, Point Q, int dir) {
  return orient(Q, u, v) == dir ? u : v;
}
Point pointPolyTangent(const Polygon &pt, Point Q, int dir, int lo, int hi) {
  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;
    bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);
    bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);

    if (pvs && nxt) return pt[mid];
    if (!(pvs || nxt)) {
      Point p1 = pointPolyTangent(pt, Q, dir, mid + 1, hi);
      Point p2 = pointPolyTangent(pt, Q, dir, lo, mid - 1);
      return better(p1, p2, Q, dir);
    }

    if (!pvs) {
      if (orient(Q, pt[mid], pt[lo]) == dir)
        hi = mid - 1;
      else if (better(pt[lo], pt[hi], Q, dir) == pt[lo])
        hi = mid - 1;
      else
        lo = mid + 1;
    }
    if (!nxt) {
      if (orient(Q, pt[mid], pt[lo]) == dir)
        lo = mid + 1;
      else if (better(pt[lo], pt[hi], Q, dir) == pt[lo])
        hi = mid - 1;
      else
        lo = mid + 1;
    }
  }

  Point ret = pt[lo];
  for (int i = lo + 1; i <= hi; i++) ret = better(ret, pt[i], Q, dir);
  return ret;
}
// [ACW, CW] Tangent
pair<Point, Point> pointPolyTangents(const Polygon &pt, Point Q) {
```

```
  int n = pt.size();
  Point acw_tan = pointPolyTangent(pt, Q, ACW, 0, n - 1);
  Point cw_tan = pointPolyTangent(pt, Q, CW, 0, n - 1);
  return make_pair(acw_tan, cw_tan);
}
```

8.4 min$_{dis_s}quares$

```
typedef long double ld;
const ld eps = 1e-12;
int cmp(ld x, ld y = 0, ld tol = eps) {
  return ( x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}
struct point{
  ld x, y;
  point(ld a, ld b) : x(a), y(b) {}
  point() {}
};
struct square{
  ld x1, x2, y1, y2, a, b, c;
  point edges[4];
  square(ld _a, ld _b, ld _c) {
    a = _a, b = _b, c = _c;
    x1 = a - c * 0.5;
    x2 = a + c * 0.5;
    y1 = b - c * 0.5;
    y2 = b + c * 0.5;
    edges[0] = point(x1, y1);
    edges[1] = point(x2, y1);
    edges[2] = point(x2, y2);
    edges[3] = point(x1, y2);
  }
};
ld min_dist(point &a, point &b) {
  ld x = a.x - b.x, y = a.y - b.y;
  return sqrt(x * x + y * y);
}
bool point_in_box(square s1, point p) {
  if (cmp(s1.x1, p.x) != 1 && cmp(s1.x2, p.x) != -1 &&
      cmp(s1.y1, p.y) != 1 && cmp(s1.y2, p.y) != -1)
      return true;
  return false;
}
bool inside(square &s1, square &s2) {
  for(int i = 0; i < 4; ++i) if(point_in_box(s2, s1.edges
      [i])) return true;
  return false;
}
bool inside_vert(square &s1, square &s2) {
  if((cmp(s1.y1, s2.y1) != -1 && cmp(s1.y1, s2.y2) != 1)
      || (cmp(s1.y2, s2.y1) != -1 && cmp(s1.y2, s2.y2) !=
       1)) return true;
  return false;
}
bool inside_hori(square &s1, square &s2) {
  if ((cmp(s1.x1, s2.x1) != -1 && cmp(s1.x1, s2.x2) != 1)
       || (cmp(s1.x2, s2.x1) != -1 && cmp(s1.x2, s2.x2)
      != 1)) return true;
  return false;
}
ld min_dist(square &s1, square &s2) {
  if (inside(s1, s2) || inside(s2, s1)) return 0;
  ld ans = 1e100;
  for (int i = 0; i < 4; ++i)
    for (int j = 0; j < 4; ++j)
      ans = min(ans, min_dist(s1.edges[i], s2.edges[j]));
  if (inside_hori(s1, s2) || inside_hori(s2, s1)) {
    if (cmp(s1.y1, s2.y2) != -1) ans = min(ans, s1.y1 -
        s2.y2);
    else if (cmp(s2.y1, s1.y2) != -1) ans = min(ans, s2.
        y1 - s1.y2);
  }
  if (inside_vert(s1, s2) || inside_vert(s2, s1)) {
    if (cmp(s1.x1, s2.x2) != -1) ans = min(ans, s1.x1 -
        s2.x2);
    else if(cmp(s2.x1, s1.x2) != -1) ans = min(ans, s2.x1
         - s1.x2);
  }
  return ans;
}
```

9 Misc

9.1 StressTest

9.2 All Macros

```
//#pragma GCC optimize("Ofast")
//#pragma GCC optimization ("O3")
//#pragma comment(linker, "/stack:200000000")
//#pragma GCC optimize("unroll-loops")
//#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm
    ,mmx,avx,tune=native")

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
    //find_by_order(k) --> returns iterator to the kth
        largest element counting from 0
    //order_of_key(val) --> returns the number of items
        in a set that are strictly smaller than our item
os.erase (os.find_by_order (os.order_of_key(v[i])))
  ==> to erase i-th element from ordered multiset
template <typename DT>
using ordered_set = tree <DT, null_type, less<DT>,
    rb_tree_tag,tree_order_statistics_node_update>;

mod = {1500000007, 1500000013, 1500000023, 1500000057,
    1500000077};

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);
  }
} Rng;

typedef gp_hash_table<int, int, custom_hash> gp;
gp table;

int leap_years(int y) { return y / 4 - y / 100 + y / 400;
     }
bool is_leap(int y) { return y % 400 == 0 || (y % 4 == 0
    && y % 100 != 0); }

bool __builtin_mul_overflow (type1 a, type2 b, type3 &res
    )
cin.tie(0)->ios_base::sync_with_stdio(0);
```

10 Equations and Formulas

10.1 Catalan Numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$ $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$

The number of ways to completely parenthesize $n+1$ factors.
The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
Number of permutations of $1, \ldots, n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.

10.2 Stirling Numbers First Kind

The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
$S(n, k)$ counts the number of permutations of n elements with k disjoint cycles.
$S(n,k) = (n-1) \cdot S(n-1,k) + S(n-1,k-1),\ where,\ S(0,0) = 1, S(n,0) = S(0,n) = 0$ $\sum_{k=0}^{n} S(n,k) = n!$
The unsigned Stirling numbers may also be defined algebraically, as the coefficient of the rising factorial:

$$x^{\bar{n}} = x(x+1)...(x+n-1) = \sum_{k=0}^{n} S(n,k)x^k$$

Lets $[n,k]$ be the stirling number of the first kind, then

$$\left[{}_{n}\overset{n}{-}{}_{k}\right] = \sum_{0 \le i_1 < i_2 < i_k < n} i_1 i_2 .... i_k.$$

10.3 Stirling Numbers Second Kind

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
$S(n,k) = k \cdot S(n-1,k) + S(n-1,k-1),\ where\ S(0,0) = 1, S(n,0) = S(0,n) = 0$ $S(n,2) = 2^{n-1} - 1$ $S(n,k) \cdot k!$ = number of ways to color n nodes using colors from 1 to k such that each color is used at least once.
An r-associated Stirling number of the second kind is the number of ways to partition a set of n objects into k subsets, with each subset containing at least r elements. It is denoted by $S_r(n,k)$ and obeys the recurrence relation. $S_r(n+1,k) = kS_r(n,k) + \binom{n}{r-1} S_r(n-r+1,k-1)$
Denote the n objects to partition by the integers $1, 2, \ldots, n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n,k)$, to be the number of ways to partition the integers $1, 2, \ldots, n$ into k nonempty subsets such that all elements in each subset have pairwise distance at least d. That is, for any integers i and j in a given subset, it is required that $|i - j| \ge d$. It has been shown that these numbers satisfy, $S^d(n,k) = S(n-d+1, k-d+1), n \ge k \ge d$

10.4 Other Combinatorial Identities

$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$

$\sum_{i=0}^{k}\binom{n+i}{i} = \sum_{i=0}^{k}\binom{n+i}{n} = \binom{n+k+1}{k}$

$n, r \in N, n > r, \sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$

If $P(n) = \sum_{k=0}^{n}\binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n}(-1)^{n-k}\binom{n}{k} \cdot P(k)$$

If $P(n) = \sum_{k=0}^{n}(-1)^k\binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n}(-1)^k\binom{n}{k} \cdot P(k)$$

10.5 Different Math Formulas

Picks Theorem : $A = i + b/2 - 1$
Deragements : $d(i) = (i-1) \times (d(i-1) + d(i-2))$

$$\frac{n}{ab} - \left\{\frac{b/n}{a}\right\} - \left\{\frac{a/n}{b}\right\} + 1$$

10.6 GCD and LCM

if m is any integer, then $\gcd(a + m \cdot b, b) = \gcd(a, b)$
The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
$\gcd(a, \operatorname{lcm}(b,c)) = \operatorname{lcm}(\gcd(a,b), \gcd(a,c))$.
$\operatorname{lcm}(a, \gcd(b,c)) = \gcd(\operatorname{lcm}(a,b), \operatorname{lcm}(a,c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$

$\gcd(a,b) = \sum_{k|a \text{ and } k|b} \phi(k)$

$\sum_{i=1}^{n}[\gcd(i,n) = k] = \phi\left(\frac{n}{k}\right)$

$\sum_{k=1}^{n}\gcd(k,n) = \sum_{d|n} d \cdot \phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$

$\sum_{k=1}^{n}\frac{1}{\gcd(k,n)} = \sum_{d|n}\frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{1}{n}\sum_{d|n} d \cdot \phi(d)$

$\sum_{k=1}^{n}\frac{k}{\gcd(k,n)} = \frac{n}{2} \cdot \sum_{d|n}\frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$

$\sum_{k=1}^{n}\frac{n}{\gcd(k,n)} = 2 * \sum_{k=1}^{n}\frac{k}{\gcd(k,n)} - 1$, for $n > 1$

$\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(i,j) = 1] = \sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor^2$

$\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j) = \sum_{d=1}^{n}\phi(d)\lfloor\frac{n}{d}\rfloor^2$

$\sum_{i=1}^{n}\sum_{j=1}^{n} i \cdot j[\gcd(i,j) = 1] = \sum_{i=1}^{n}\phi(i)i^2$

$F(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\operatorname{lcm}(i,j) = \sum_{l=1}^{n}\left(\frac{\left(1 + \lfloor\frac{n}{l}\rfloor\right)\left(\lfloor\frac{n}{l}\rfloor\right)}{2}\right)^2 \sum_{d|l}\mu(d)ld$