Contents

1   Build and Snippet

1.1   Sublime Build

```
{
"cmd" : ["g++ -std=c++20 -DLOCAL -Wall $file_name -o
$file_base_name && timeout 5s ./$file_base_name<input.
txt> output.txt && rm $file_base_name"],
"selector" : "source.cpp",
"file_regex": "(..[ :]*):([0-9]+):?([0-9]+)?:? (.*)$",
"shell": true,
"working_dir" : "$file_path"
}
{
    "shell_cmd": "g++ -std=c++23 \"$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++"
}// interactive
"shell_cmd": "g++ -std=c++23 \"$file\" -o \"
$file_base_name.exe\" && start cmd /k \"$file_base_name.
exe\"",
```

2   Data Structures

2.1   2D BIT

```cpp
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

```cpp
class BIT {
```

```cpp
int *bin, N;
public:
BIT(int n) : N(n + 1) {
  bin = new int[N + 1];
  memset(bin, 0, (N + 1) * sizeof(int));
}
void update(int id, int val) {
  id++;
  for (; id <= N; id += id & -id) bin[id] += val;
}
int helper(int id) {
  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   Beats

```cpp
class SegTreeBeats {
  const int INF = INT_MAX;
  const LL NEG_INF = LLONG_MIN;

  vector<LL> mx, mn, smx, smn, sum, add;
  vector<int> mxcnt, mncnt;
  int L, R;

  void applyMax(int u, LL x) {
    sum[u] += mncnt[u] * (x - mn[u]);
    if (mx[u] == mn[u]) mx[u] = x;
    if (smx[u] == mn[u]) smx[u] = x;
    mn[u] = x;
  }

  void applyMin(int u, LL x) {
    sum[u] -= mxcnt[u] * (mx[u] - x);
    if (mn[u] == mx[u]) mn[u] = x;
    if (smn[u] == mx[u]) smn[u] = x;
    mx[u] = x;
  }

  void applyAdd(int u, LL x, int tl, int tr) {
    sum[u] += (tr - tl + 1) * x;
```

```cpp
    add[u] += x;
    mx[u] += x, mn[u] += x;
    if (smx[u] != NEG_INF) smx[u] += x;
    if (smn[u] != INF) smn[u] += x;
  }

  void push(int u, int tl, int tr) {
    int lft = u << 1, ryt = lft | 1, mid = (tl + tr) >>
    1;
    if (add[u] != 0) {
      applyAdd(lft, add[u], tl, mid);
      applyAdd(ryt, add[u], mid + 1, tr);
      add[u] = 0;
    }
    if (mx[u] < mx[lft]) applyMin(lft, mx[u]);
    if (mx[u] < mx[ryt]) applyMin(ryt, mx[u]);

    if (mn[u] > mn[lft]) applyMax(lft, mn[u]);
    if (mn[u] > mn[ryt]) applyMax(ryt, mn[u]);
  }

  void merge(int u) {
    int lft = u << 1, ryt = lft | 1;
    sum[u] = sum[lft] + sum[ryt];

    mx[u] = max(mx[lft], mx[ryt]);
    smx[u] = max(smx[lft], smx[ryt]);
    if (mx[lft] != mx[ryt]) smx[u] = max(smx[u], min(mx[
    lft], mx[ryt]));
    mxcnt[u] =
        (mx[u] == mx[lft]) * mxcnt[lft] + (mx[u] == mx[
    ryt]) * mxcnt[ryt];

    mn[u] = min(mn[lft], mn[ryt]);
    smn[u] = min(smn[lft], smn[ryt]);
    if (mn[lft] != mn[ryt]) smn[u] = min(smn[u], max(mn[
    lft], mn[ryt]));
    mncnt[u] =
        (mn[u] == mn[lft]) * mncnt[lft] + (mn[u] == mn[
    ryt]) * mncnt[ryt];
  }

  void build(const vector<int>& a, int tl, int tr, int u
  ) {
    if (tl == tr) {
      sum[u] = mn[u] = mx[u] = a[tl];
```

```cpp
      mxcnt[u] = mncnt[u] = 1;
      smx[u] = NEG_INF;
      smn[u] = INF;
      return;
    }
    int mid = (tl + tr) >> 1, lft = u << 1, ryt = lft |
    1;
    build(a, tl, mid, lft);
    build(a, mid + 1, tr, ryt);
    merge(u);
  }

public:
  SegTreeBeats(const vector<int>& a) {
    int n = a.size();
    L = 0;
    R = n - 1;

    mx.resize(4 * n, 0);
    mn.resize(4 * n, 0);
    smx.resize(4 * n, NEG_INF);
    smn.resize(4 * n, INF);
    sum.resize(4 * n, 0);
    add.resize(4 * n, 0);
    mxcnt.resize(4 * n, 0);
    mncnt.resize(4 * n, 0);

    build(a, L, R, 1);
  }
  // a[i] = min(x, a[i]);
  void minimize(int l, int r, LL x) { minimize(l, r, x,
  L, R, 1); }
  // a[i] = max(x, a[i]);
  void maximize(int l, int r, LL x) { maximize(l, r, x,
  L, R, 1); }
  // a[i] = a[i] + x;
  void increase(int l, int r, LL x) { increase(l, r, x,
  L, R, 1); }

  LL getSum(int l, int r) { return getSum(l, r, L, R, 1)
  ; }

  LL getMax(int l, int r) { return getMax(l, r, L, R, 1)
  ; }

private:
```

```cpp
  void minimize(int l, int r, LL x, int tl, int tr, int
  u) {
    if (l > tr || tl > r || mx[u] <= x) return;
    if (l <= tl && tr <= r && smx[u] < x) {
      applyMin(u, x);
      return;
    }
    push(u, tl, tr);
    int mid = (tl + tr) >> 1, lft = u << 1, ryt = lft |
    1;
    minimize(l, r, x, tl, mid, lft);
    minimize(l, r, x, mid + 1, tr, ryt);
    merge(u);
  }

  void maximize(int l, int r, LL x, int tl, int tr, int
  u) {
    if (l > tr || tl > r || mn[u] >= x) return;
    if (l <= tl && tr <= r && smn[u] > x) {
      applyMax(u, x);
      return;
    }
    push(u, tl, tr);
    int mid = (tl + tr) >> 1, lft = u << 1, ryt = lft |
    1;
    maximize(l, r, x, tl, mid, lft);
    maximize(l, r, x, mid + 1, tr, ryt);
    merge(u);
  }

  void increase(int l, int r, LL x, int tl, int tr, int
  u) {
    if (l > tr || tl > r) return;
    if (l <= tl && tr <= r) {
      applyAdd(u, x, tl, tr);
      return;
    }
    push(u, tl, tr);
    int mid = (tl + tr) >> 1, lft = u << 1, ryt = lft |
    1;
    increase(l, r, x, tl, mid, lft);
    increase(l, r, x, mid + 1, tr, ryt);
    merge(u);
  }

  LL getSum(int l, int r, int tl, int tr, int u) {
```

```cpp
    if (l > tr || tl > r) return 0;
    if (l <= tl && tr <= r) return sum[u];
    push(u, tl, tr);
    int mid = (tl + tr) >> 1, lft = u << 1, ryt = lft |
    1;
    return getSum(l, r, tl, mid, lft) + getSum(l, r, mid
     + 1, tr, ryt);
  }

  LL getMax(int l, int r, int tl, int tr, int u) {
    if (l > tr || tl > r) return NEG_INF;
    if (l <= tl && tr <= r) return mx[u];
    push(u, tl, tr);
    int mid = (tl + tr) >> 1;
    return max(getMax(l, r, tl, mid, u << 1),
               getMax(l, r, mid + 1, tr, (u << 1) | 1));
  }
};
```

2.4 Implicit PST

```cpp
class PST{
    struct node{
        int left = 0, right = 0, val = 0;
    };
    vector<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 update(int id, int le, int ri, int pos, int val)
    {
        int nid = ++time;
        if(le == ri) return tree.push_back({0, 0, (tree[
        id].val ^ val)}), nid;
        int m = (le + ri) / 2;
        tree.push_back(node());
        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 -1;
        if(le == ri) return le;
        int m = (le + ri) >> 1;

        if(tree[tree[id].right].val != tree[tree[di].
        right].val)
            return query(tree[id].right, tree[di].right,
            m + 1, ri);
        return query(tree[id].left, tree[di].left, le, m)
        ;
    }
    int merge(LL a, LL b){
        return a ^ b;
    }
public:
    PST(int N){
        this->N = N;
        tree.push_back(node());
    }
    int update(int id, int pos, int val){
        return update(id, 1, N, pos, val);
    }
    int query(int id, int di){
        return query(id, di, 1, N);
    }
    ~PST(){
    }
};
```

2.5   Lazy Propagation

```
template <typename T, typename U>
struct LazySegmentTree {
  vector<T> tree;
  vector<U> lazy;
  int n;
  T I;
  function<T(T, T)> merge;
  void updateLazy(int id, U val) { lazy[id] += val; }
  void push(int id, int le, int ri) {
    tree[id] += (ri - le + 1) * lazy[id];
    if (le != ri) {
      updateLazy(2 * id + 1, lazy[id]);
      updateLazy(2 * id + 2, lazy[id]);
```

```
    }
    lazy[id] = 0;
}
void fix(int id) { tree[id] = merge(tree[2 * id + 1],
tree[2 * id + 2]); }
void build(int id, int le, int ri, vector<T>& arr) {
    if (le == ri) {
      tree[id] = arr[le];
      return;
    }
    int mid = (le + ri) / 2;
    build(2 * id + 1, le, mid, arr);
    build(2 * id + 2, mid + 1, ri, arr);
    fix(id);
}
void update(int id, int le, int ri, int l, int r, U
val) {
    push(id, le, ri);
    if (r < le || l > ri) return;
    if (le >= l && ri <= r) {
      updateLazy(id, val), push(id, le, ri);
      return;
    }
    int mid = (le + ri) / 2;
    update(2 * id + 1, le, mid, l, r, val);
    update(2 * id + 2, mid + 1, ri, l, r, val);
    fix(id);
}
T query(int id, int le, int ri, int l, int r) {
    push(id, le, ri);
    if (r < le || l > ri) return I;
    if (l <= le && ri <= r) return tree[id];
    int mid = (le + ri) / 2;
    return merge(query(2 * id + 1, le, mid, l, r),
                 query(2 * id + 2, mid + 1, ri, l, r));
}
int findL(int id, int le, int ri, int l, int r, T val)
 {
    push(id, le, ri);
    if (r < le || l > ri) return r + 1;
    if (l <= le && ri <= r && tree[id] > val) return r +
     1;
    if (le == ri) return le;

    int mid = (le + ri) / 2;
    int left = findL(2 * id + 1, le, mid, l, r, val);
```

```
    if (left <= r) return left;
    return findL(2 * id + 2, mid + 1, ri, l, r, val);
  }
 public:
  LazySegmentTree(vector<T>& arr, T I, function<T(T, T)>
   merge)
      : n(arr.size()), I(I), merge(merge) {
    tree.assign(4 * n, I);
    lazy.assign(4 * n, 0);
    build(0, 0, n - 1, arr);
  }
  LazySegmentTree(int n, T I, function<T(T, T)> merge)
      : n(n), I(I), merge(merge) {
    tree.assign(4 * n, I);
    lazy.assign(4 * n, 0);
  }
  void update(int l, int r, U val) { update(0, 0, n - 1,
   l, r, val); }
  T query(int l, int r) { return query(0, 0, n - 1, l, r
  ); }
  int findL(int l, int r, T val) { return findL(0, 0, n
  - 1, l, r, val); }
};
```

2.6   MO With Update

```
const int N = 1e5 + 5, sz = 2700, bs = 25;
int arr[N], freq[2 * N], cnt[2 * N], id[N], ans[N];
struct query {
  int l, r, t, L, R;
  query(int l = 1, int r = 0, int t = 1, int id = -1)
      : l(l), r(r), t(t), L(l / sz), R(r / sz) {}
  bool operator<(const query& rhs) const {
    return (L < rhs.L) or (L == rhs.L and R < rhs.R) or
           (L == rhs.L and R == rhs.R and t < rhs.t);
  }
} Q[N];
struct update {
  int idx, val, last;
} Up[N];
int qi = 0, ui = 0;
int l = 1, r = 0, t = 0;
void add(int idx) {
  --cnt[freq[arr[idx]]];
  freq[arr[idx]]++;
  cnt[freq[arr[idx]]]++;
}
```

```cpp
void remove(int idx) {
  --cnt[freq[arr[idx]]];
  freq[arr[idx]]--;
  cnt[freq[arr[idx]]]++;
}
void apply(int t) {
  const bool f = l <= Up[t].idx and Up[t].idx <= r;
  if (f) remove(Up[t].idx);
  arr[Up[t].idx] = Up[t].val;
  if (f) add(Up[t].idx);
}
void undo(int t) {
  const bool f = l <= Up[t].idx and Up[t].idx <= r;
  if (f) remove(Up[t].idx);
  arr[Up[t].idx] = Up[t].last;
  if (f) add(Up[t].idx);
}
int mex() {
  for (int i = 1; i <= N; i++)
    if (!cnt[i]) return i;
  assert(0);
}
int main() {
  int n, q;
  cin >> n >> q;
  int counter = 0;
  map<int, int> M;
  for (int i = 1; i <= n; i++) {
    cin >> arr[i];
    if (!M[arr[i]]) M[arr[i]] = ++counter;
    arr[i] = M[arr[i]];
  }
  iota(id, id + N, 0);
  while (q--) {
    int tp, x, y;
    cin >> tp >> x >> y;
    if (tp == 1)
      Q[++qi] = query(x, y, ui);
    else {
      if (!M[y]) M[y] = ++counter;
      y = M[y];
      Up[++ui] = {x, y, arr[x]};
      arr[x] = y;
    }
  }
  t = ui;
```

```cpp
  cnt[0] = 3 * n;
  sort(id + 1, id + qi + 1, [&](int x, int y) { return Q
[x] < Q[y]; });
  for (int i = 1; i <= qi; i++) {
    int x = id[i];
    while (Q[x].t > t) apply(++t);
    while (Q[x].t < t) undo(t--);
    while (Q[x].l < l) add(--l);
    while (Q[x].r > r) add(++r);
    while (Q[x].l > l) remove(l++);
    while (Q[x].r < r) remove(r--);
    ans[x] = mex();
  }
  for (int i = 1; i <= qi; i++) cout << ans[i] << '\n';
}
```

2.7 MO

```cpp
struct node { LL l, r, idx; };
bool cmp(const node &x, const node &y) {
  return x.r < y.r;
}
LL curr = 0, BLOCK_SIZE, m;
vector<node> query[M];
vector<LL> mp(N, 0);
void add(LL x) {
  if(!mp[x]) curr++;
  mp[x]++;
}
void diminish(LL x) {
  mp[x]--;
  if(!mp[x]) curr--;
}
void solve() {
  int n, q; cin >> n >> q;
  vector<int> v(n), answer(q);
  for(int &i : v) cin >> i;
  BLOCK_SIZE = sqrt(n) + 1, m = 0;
  for(int i=0;i<q;i++) {
    LL x, y; cin >> x >> y; x--; y--;
    query[x / BLOCK_SIZE].push_back({x, y, i});
    m = max(m, x / BLOCK_SIZE);
  }
  for(int i=0;i<=m;i++) sort(all(query[i]), cmp);
  LL mo_left = 0, mo_right = -1;
  for(int i=0;i<=m;i++) {
    for(auto cur : query[i]) {
```

```cpp
      LL left = cur.l, right = cur.r, id = cur.idx;
      while(mo_right < right) add(v[++mo_right]);
      while(mo_right > right) diminish(v[mo_right--]);
      while(mo_left < left) diminish(v[mo_left++]);
      while(mo_left > left) add(v[--mo_left]);
      answer[id] = curr;
    }
  }
  for(int i=0;i<q;i++) cout << answer[i] << '\n';
}
```

2.8 PST

```cpp
// 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] =
```

```cpp
        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 * 4 + 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; }
};
```

2.9   Sparse Table

```cpp
class SparsedTable {
private:
  vector<vector<int>> table;
  vector<int> log;
  int n;
public:
  SparsedTable(const vector<int>& arr) {
    n = arr.size();
    log.resize(n + 1);
    buildLog();
    table = vector<vector<int>>(log[n] + 1, vector<int>(
    n));
```

```cpp
    for (int i = 0; i < n; i++) {
      table[0][i] = arr[i];
    }
    for (int j = 1; j <= log[n]; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        table[j][i] = merge(table[j - 1][i], table[j -
        1][i + (1 << (j - 1))]);
      }
    }
  }
  int merge(int a, int b) { return 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[j][L], table[j][R - (1 << j) +
    1]);
  }
  void update(int L, int R, int val) {
    int j = log[R - L + 1];
    table[j][L] = min(table[j][L], val);
    table[j][R - (1 << j) + 1] = min(table[j][R - (1 <<
    j) + 1], val);
  }
  void finalize() {
    for (int j = log[n]; j > 0; j--) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        table[j - 1][i] = min(table[j - 1][i], table[j][i
        ]);
        table[j - 1][i + (1 << (j - 1))] =
            min(table[j - 1][i + (1 << (j - 1))], table[j
            ][i]);
      }
    }
  }
};
```

2.10   Trie

```cpp
struct node {
  int path, leaf;
  vector<int> child;
  node(int n = 0) : child(n, -1), path(0), leaf(0) {}
```

```cpp
};
class Trie {
  int n, ptr;
  vector<node> tree;
public:
  Trie(int n) : n(n), ptr(0) { tree.emplace_back(node(n)
  ); }
  void insert(string& s) {
    int cur = 0;
    for (auto u : s) {
      int& next = tree[cur].child[u - '0'];
      if (next == -1) {
        tree.emplace_back(node(n));
        next = ++ptr;
      }
      tree[cur].path++; cur = next;
    }
    tree[cur].path++; tree[cur].leaf++;
  }
  void erase(string& s) {
    int cur = 0;
    for (auto u : s) {
      tree[cur].path--;
      cur = tree[cur].child[u - '0'];
    }
    tree[cur].path--; tree[cur].leaf--;
  }
  bool find(string& s) {
    int cur = 0;
    for (auto u : s) {
      cur = tree[cur].child[u - '0'];
      if (cur == -1 || !tree[cur].path) return 0;
    }
    return tree[cur].leaf > 0;
  }
};
```

2.11   segtree iterative

```cpp
template <class T, class L, T it, T (*mergeT)(T, T), T
(*applyL)(T, L)>
struct SegTree {
  int n;
  vector<T> tree;
  void update(int pos, L val) {
    pos += n;
    tree[pos] = applyL(tree[pos], val);
```

```cpp
    while (pos > 1) {
      pos >>= 1;
      tree[pos] = mergeT(tree[pos << 1], tree[pos << 1 |
      1]);
    }
  }
  T query(int l, int r) {
    l += n, r += n + 1;
    T lft = it, ryt = it;
    while (l < r) {
      if (l & 1) lft = mergeT(lft, tree[l++]);
      if (r & 1) ryt = mergeT(tree[--r], ryt);
      l >>= 1;
      r >>= 1;
    }
    return mergeT(lft, ryt);
  }
  SegTree(vector<T>& v) : n(v.size()) {
    tree.assign(2 * n, it);

    for (int i = 0; i < n; i++) tree[i + n] = v[i];
    for (int i = n - 1; i > 0; i--)
      tree[i] = mergeT(tree[i << 1], tree[i << 1 | 1]);
  }
};
```

2.12 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) {
    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]);
    } } }
```

```cpp
  }
  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 Chinese Reminder Theorem

```cpp
// 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);
}
```

3.2 FFT

```cpp
// typedef complex<double> CD;
struct CD {
  double x, y;
  CD(double x = 0, double y = 0) : x(x), y(y) {}
```

```cpp
  CD operator+(const CD &o) { return {x + o.x, y + o.y};
   }
  CD operator-(const CD &o) { return {x - o.x, y - o.y};
   }
  CD operator*(const CD &o) { return {x * o.x - y * o.y,
   x * o.y + o.x * y}; }
  void operator/=(double d) { x /= d; y /= d; }
  double real() { return x; }
  double imag() { return y; }
};
CD conj(const CD &c) { return CD(c.x, -c.y); }
typedef long long LL;
const double PI = acos(-1.0L);
namespace FFT {
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; }
}
```

```cpp
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<bool> multiply(const vector<bool> &a, const
vector<bool> &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<bool> ans(n);
  for (int i = 0; i < n; i++) ans[i] = round(fa[i].real
  ());
  while (ans.size() > 1 && !ans.back()) ans.pop_back();
  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);
```

```cpp
  //       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;
}
}  // namespace FFT
```

3.3 Matrix

```cpp
template <typename DT>
struct Matrix {
  vector<vector<DT>> mat;
  Matrix(vector<vector<DT>>& mat) : mat(mat) {}
  Matrix(int n) : mat(n, vector<DT>(n)) {
    for (int i = 0; i < n; i++) {
      mat[i][i] = 1;
} }
  Matrix(int n, int m) : mat(n, vector<DT>(m)) {}
  Matrix operator*(Matrix& other) {
    auto &a = mat, &b = other.mat;
    assert(a[0].size() == b.size());
    int n = a.size(), m = b[0].size(), s = a[0].size();
    Matrix<DT> ret(n, m);
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        DT temp = 0;
        for (int k = 0; k < s; k++) {
          temp = (temp + 1LL * a[i][k] * b[k][j] % MOD) %
           MOD;
        }
```

```cpp
        ret.mat[i][j] = temp;
    } }
    return ret;
} };
```

3.4 Mint

```cpp
class mint {
private:
    long long value;
    static int M;
    void normalize() {
        value %= M;
        if (value < 0) value += M;
    }
public:
    mint() : value(0) {}
    mint(int value) : value(value) { normalize(); }
    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 % M); }
    mint operator*(int other) const { return mint(value
* other % M); }
    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; }
```

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

  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 mint::M = mod;
```

3.5 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);
}
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
```

3.6 Pollard rho

```cpp
namespace rho {
```

```cpp
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;
}
// LL bigmod(LL num, LL pow, LL mod)
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,
```

```cpp
  83, 89, 97};
  for (LL p : small_primes)
    for (; n % p == 0; n /= p, res[p]++);
  _factor(n, res);
  return res;
} } // namespace rho
```

3.7 Sieve

```cpp
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;
  }
} }
```

3.8 XOR Basis

```cpp
template <typename T = int, int B = 31>
struct Basis {
  T a[B];
  Basis() { memset(a, 0, sizeof a); }
  void insert(T x) {
    for (int i = B - 1; i >= 0; i--) {
      if (x >> i & 1) {
        if (a[i]) x ^= a[i];
        else { a[i] = x; break; }
  } } }
  bool can(T x) {
    for (int i = B - 1; i >= 0; i--)
      x = min(x, x ^ a[i]);
    return x == 0;
  }
  T max_xor(T ans = 0) {
    for (int i = B - 1; i >= 0; i--)
      ans = max(ans, ans ^ a[i]);
    return ans;
  }
} };
```

3.9 linear diophantine

```cpp
using PLL = pair <LL,LL>;
// EGCD
tuple <bool, LL, LL, LL> get_solution(LL a, LL b, LL c)
{
```

```cpp
auto [x, y, g] = EGCD(abs(a), abs(b));
x *= (a < 0) ? -c / g : c / g;
y *= (b < 0) ? -c / g : c / g;
return {c % g == 0, x, y, g} ; }
PLL shift_solution(LL a, LL b, LL x, LL y, LL shift) {
return {x + shift * b, y - shift * a}; }
```

3.10   mobius function

```cpp
const int N = 1e6 + 5;
int mob[N];
void mobius() {
  memset(mob, -1, sizeof mob);
  mob[1] = 1;
  for (int i = 2; i < N; i++)
    if (mob[i]) {
      for (int j = i + i; j < N; j += i) mob[j] -= mob[i
      ];
    }
} }
```

3.11   nCr

```cpp
namespace com {
LL fact[N], inv[N], inv_fact[N];
void init() {
  fact[0] = inv_fact[0] = 1;
  for (LL 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
```

3.12   ntt

```cpp
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 =
primitive_root (MOD);
// LL Pow = bigmod
LL primitive_root(LL p) {
  vector<LL> factor;
  LL phi = p - 1, n = phi;
  for (LL i = 2; i * i <= n; i++) {
```

```cpp
    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> multiply (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;
```

```cpp
  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;
}
```

3.13   primality test

```cpp
using u64 = uint64_t;
using u128 = __uint128_t;
// u64 binpower(u64 base, u64 e, u64 mod)
bool check_composite(u64 n, u64 a, u64 d, int s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (int r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1) return false;
  }
  return true;
};
bool MillerRabin(u64 n, int iter = 5) { // returns true
if n is probably prime, else returns false.
  if (n < 4) return n == 2 || n == 3;
  int s = 0; u64 d = n - 1;
  while ((d & 1) == 0) d >>= 1, s++;
  for (int i = 0; i < iter; i++) {
    int a = 2 + rand() % (n - 3);
    if (check_composite(n, a, d, s)) return false;
  } return true;
}
```

4   Graph

4.1   BridgeTree

```cpp
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;
    }
```

```cpp
    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]);
    }
  }
}
```

4.2 CD

```cpp
class CD {
  vector<vector<int>> adj;
  vector<int> sub;
```

```cpp
  vector<bool> blocked;
  int N;
public:
  CD(vector<vector<int>> adj) : adj(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;
```

```cpp
  }
};
```

4.3 DSU with rollback

```cpp
struct Rollback_DSU {
  int n;
  vector <int> par, sz;
  vector <pair <int, int>> op;
  Rollback_DSU(int n) : par(n), sz(n, 1) {
    iota(par.begin(), par.end(), 0);
    op.reserve(n);
  }
  int Anc(int node) {
    for(; node != par[node]; node = par[node]);
    return node;
  }
  bool Unite(int x, int y) {
    x = Anc(x), y = Anc (y);
    if (x == y) return false;
    if(sz[x] < sz[y]) swap(x, y);
    op.emplace_back(x, y);
    par[y] = x;
    sz[x] += sz[y];
    return true;
  }
  void Undo(size_t t) {
    for(; op.size() > t; op.pop_back()) {
      par[op.back().second] = op.back().second;
      sz[op.back().first] -= sz[op.back().second];
    }
  }
};
```

4.4 DSU, MST

```cpp
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); }
```

```cpp
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
```

4.5 Dinic

```cpp
struct Dinic {
  struct edge {
    int to, rid, isRev;
    LL cap, flow;
  };
  int n, s, t;
  vector<vector<edge>> g;
  vector<int> level, ptr;
  Dinic(int _n) : n(_n + 5) { g.resize(n); }
  void addEdge(int u, int v, LL cap) {
    g[u].push_back({v, (int)g[v].size(), 0, cap, OLL});
    g[v].push_back({u, (int)g[u].size() - 1, 1, OLL, OLL
    });
  }
  bool bfs() {
    level.assign(n, -1);
    queue<int> q;
    q.push(s);
    level[s] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto& e : g[u]) {
```

```cpp
        int v = e.to;
        if (level[v] == -1 and e.cap > e.flow)
          level[v] = level[u] + 1, q.push(v);
      }
    }
    return level[t] != -1;
  }
  LL dfs(int u, LL flow) {
    if (u == t) return flow;
    for (int& i = ptr[u]; i < (int)g[u].size(); i++) {
      edge& e = g[u][i];
      int v = e.to;
      if (level[v] != level[u] + 1 or e.cap <= e.flow)
      continue;
      LL cflow = dfs(v, min(flow, e.cap - e.flow));
      if (cflow) {
        e.flow += cflow, g[v][e.rid].flow -= cflow;
        return cflow;
      }
    }
    return 0;
  }
  LL maxFlow(int s, int t) {
    this->s = s, this->t = t;
    LL flow = 0;
    while (bfs()) {
      ptr.assign(n, 0);
      while (LL cflow = dfs(s, INF)) flow += cflow;
    }
    return flow;
  }
};
```

4.6 HLD

```cpp
const int N = 2E5 + 69;
int n;
// seg tree here
vector<int> adj[N];
int cur_pos;
bool WEIGHT_IN_EDGE = false;
int Parent[N], Depth[N], Heavy[N], Head[N], Pos[N];
int Start[N], End[N]; vector<int> Euler; /// ONLY
NEEDED FOR SUBTREE QUERIES
void reset (bool WIE = false) {
  cur_pos = 0;
  WEIGHT_IN_EDGE = WIE;
```

```cpp
  fill (Heavy, Heavy + n, -1);
  // build (vector <T> (n, I));
}
int dfs (int v = 0) {
  int size = 1, max_c_size = 0;
  for (int c : adj[v]) {
    if (c != Parent[v]) {
      Parent[c] = v;
      Depth[c] = Depth[v] + 1;
      int c_size = dfs(c);
      size += c_size;
      if (c_size > max_c_size) {
        max_c_size = c_size;
        Heavy[v] = c;
  } } }
  return size;
}
void decompose (int v = 0, int h = 0) {
  Head[v] = h; Pos[v] = cur_pos++;
  Euler.push_back(v);
  Start[v] = (int)Euler.size() - 1;
  if (Heavy[v] != -1) decompose(Heavy[v], h);
  for (int c : adj[v]) if (c != Parent[v] && c != Heavy[
  v]) decompose(c, c);
  End[v] = Euler.size() - 1;
}
void update_path (int a, int b, LL val) {
  while (Head[a] != Head[b]) {
    if (Depth[Head[a]] < Depth[Head[b]]) swap(a, b);
    update(Pos[Head[a]], Pos[a], val);
    a = Parent[Head[a]];
  }
  if (Depth[a] > Depth[b]) swap(a, b);
  update(Pos[a] + WEIGHT_IN_EDGE, Pos[b], val);
}
// LL query_path (int a, int b) {
//   LL res = 0;
//   while (Head[a] != Head[b]) {
//     if (Depth[Head[a]] < Depth[Head[b]]) swap(a, b);
//     res = merge (res, query(Pos[Head[a]], Pos[a]));
//     a = Parent[Head[a]];
//   }
//   if (Depth[a] > Depth[b]) swap(a, b);
//   res = merge (res, query(Pos[a] + WEIGHT_IN_EDGE, Pos
[b]));
//   return res;
```

```cpp
// }
vector <T> query_up(int u, int anc) {
  vector <T> res;
  while (Head[u] != Head[anc]) {
    res.emplace_back (query(Pos[Head[u]], Pos[u]));
    u = Parent[Head[u]];
  }
  res.emplace_back (query(Pos[anc] + 1, Pos[u]));
  return res;
}
LL query_path(int u, int v) {
  int w = LCA:: lca(u, v);
  auto left = query_up(u, w);
  auto right = query_up(v, w);
  node res = I;
  for (auto e : left) res = merge (e, res);
  if (not WEIGHT_IN_EDGE) res = merge(query(Pos[w], Pos[
  w]), res);
  swap (res.AP, res.PA);
  auto res2 = I;
  for (auto e : right) res2 = merge (e, res2);
  res = merge (res, res2);
  return max ({res.PP, res.PA, res.AP, res.AA});
}
void update_subtree (int v, LL val) { update(Start[v],
End[v], val); }
LL query_subtree (int v) { return query(Start[v], End[v
]); }
void HLD_initialize (vector<LL> &weight) {
  reset (true); Parent[0] = -1;
  dfs(); decompose();
  for (int i = 0; i < n; i++) update(Pos[i], Pos[i],
  weight[i]);
}
```

4.7   LCA O(1)

```cpp
void dfs(int u, int p, int d, vector<int> g[], vector<
pair<int, int>> &order, vector<int> &mp){
    for(auto v: g[u]){
        if(v == p) continue;
        order.push_back({d, u});
        dfs(v, u, d + 1, g, order, mp);
    }
    order.push_back({d, u});
    mp[u] = (int) order.size() - 1;
}
```

```cpp
void solve(int tc) {
    vector<int> g[n + 1], mp(n + 1);
    vector<pair<int, int>> order;
    dfs(1, 0, 0, g, order, mp);
    SparsedTable<pair<int, int>> sp(order);
    auto lca = [&](int u, int v){
        if(mp[u] > mp[v]) swap(u, v);

        return sp.query(mp[u], mp[v]).second;
    };
    auto dis = [&](int u, int v){
        int w = lca(u, v);
        u = mp[u], v = mp[v], w = mp[w];
        return order[u].first + order[v].first - 2 *
        order[w].first;
    };
}
```

4.8   LCA O(n)

```cpp
struct LCA {
  int n;
  vector<int> lvl, par, jmp;
  LCA(vector<vector<int>> const& g, int u, int p) : n(g.
  size()), lvl(n, -1), par(n, -1), jmp(n, p) {
    lvl[u] = 0, par[u] = p;
    queue<int> q;
    for (q.push(u); q.size(); q.pop()) {
      int u = q.front();
      for (int v : g[u])
        if (lvl[v] < 0) {          q.push(v);
          lvl[v] = lvl[u] + 1, par[v] = u;
          if (lvl[jmp[u]] << 1 == lvl[u] + lvl[jmp[jmp[u
          ]]]) jmp[v] = jmp[jmp[u]];
          else jmp[v] = u;
        }
    }
  }
  int getAnc(int u, int d) {
    d = max(0, lvl[u] - d);
    while (lvl[u] > d) {
      if (lvl[jmp[u]] < d) u = par[u];
      else u = jmp[u];
    }
    return u;
  }
```

```cpp
  int lca(int u, int v) {
    if (lvl[u] < lvl[v]) swap(u, v);
    u = getAnc(u, lvl[u] - lvl[v]);
    while (u != v) {
      if (jmp[u] == jmp[v]) u = par[u], v = par[v];
      else u = jmp[u], v = jmp[v];
    }
    return u;
  }
  int dist(int u, int v) { return lvl[u] + lvl[v] - 2 *
  lvl[lca(u, v)]; }
};
```

4.9   block cut tree

```cpp
const int N = 4e5 + 1;
bitset<N> art;
vector<int> g[N], tree[N], st, comp[N];
int n, ptr, cur, in[N], low[N], id[N];
void memclear() {
  for (int i = 1; i <= n; i++) {
    g[i].clear(); in[i] = low[i] = art[i] = 0;
  }
  for (int i = 1; i <= ptr; i++) {
    tree[i].clear(); comp[i].clear();
  }
  st.clear();
  ptr = cur = 0;
}
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] = ~from or in[v] > in[u] + 1;
          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 build_tree() {
```

```
  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;
    } }
}
```

4.10 maximum bipartite matching hopkroft

```
// do everything 1-based
class BipartiteMatcher {
private:
  int n, m;             // Number of vertices in left
  and right sets
  vector<vector<int>> adj; // Adjacency list for the
  bipartite graph
  vector<int> dist;     // Distance array for BFS
  vector<int> matchL, matchR;
  // matchL[u] is the right vertex matched with u;
  // matchR[v] is the left vertex matched with v
public:
  BipartiteMatcher(int n, int m) : n(n), m(m) {
    adj.resize(n + 1);
    matchL.resize(n + 1, 0);
    matchR.resize(m + 1, 0);
    dist.resize(n + 1);
  }
  void addEdge(int u, int v) { adj[u].push_back(v); }
  bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
      if (matchL[u] == 0) {
        dist[u] = 0;
        q.push(u);
      } else {
        dist[u] = INT_MAX;
      }
    }
    dist[0] = INT_MAX;
    while (!q.empty()) {
      int u = q.front(); q.pop();
```

```
      if (dist[u] < dist[0]) {
        for (int v : adj[u]) {
          if (dist[matchR[v]] == INT_MAX) {
            dist[matchR[v]] = dist[u] + 1;
            q.push(matchR[v]);
          }
        }
      }
    }
    return dist[0] != INT_MAX;
  }
  bool dfs(int u) {
    if (u != 0) {
      for (int v : adj[u]) {
        if (dist[matchR[v]] == dist[u] + 1 and dfs(matchR
        [v])) {
          matchL[u] = v; matchR[v] = u;
          return true;
        }
      }
      dist[u] = INT_MAX;
      return false;
    }
    return true;
  }
  int hopcroftKarp() {
    int matching = 0;
    while (bfs())
      for (int u = 1; u <= n; u++)
        if (matchL[u] == 0 && dfs(u)) matching++;
    return matching;
  }
};
```

4.11 strongly connected component

```
bool vis[N];
vector<int> adj[N], adjr[N];
vector<int> order, component;
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();
    }
  }
}
```

5 String

5.1 Aho

```
const int N = 2000, L = 105;
struct AhoCorasick {
  int N, P;
  const int A = 256;
  vector<vector<int>> next;
  vector<int> link, out_link, end_in_pattern;
  vector<vector<int>> out;
  AhoCorasick() : N(0), P(0) { node(); }
  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    end_in_pattern.emplace_back(0);
    return N++;
  }
  inline int get(char c) { return c; }
  int addPattern(const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
    }
    out[u].push_back(P);
    end_in_pattern[u] = 1;
    return P++;
  }
  void pushLinks() {
    queue<int> q;
    for (q.push(0); !q.empty();) {
      int u = q.front();
```

```cpp
    q.pop();
    for (int c = 0; c < A; ++c) {
      int v = next[u][c];
      if (!v) next[u][c] = next[link[u]][c];
      else {
        link[v] = u ? next[link[u]][c] : 0;
        out_link[v] = out[link[v]].empty() ? out_link[
          link[v]] : link[v];
        q.push(v);
      }
      end_in_pattern[v] |= end_in_pattern[out_link[v]];
  } } }
  int advance(int u, char c) {
    while (u && !next[u][get(c)]) u = link[u];
    u = next[u][get(c)]; return u;
} };
```

5.2 KMP

```cpp
template <typename T>
vector<int> getLPSarray(T& b) {
  int m = b.size();
  vector<int> phi(m);
  for (int i = 1, j = 0; i < m; i++) {
    while (j && b[i] != b[j]) j = phi[j - 1];
    if (b[i] == b[j]) j++;
    phi[i] = j;
  } return phi;
}
template <typename T>
int KMP_match(T& a, T& b) {
  int n = a.size(), m = b.size();
  vector<int> lps = getLPSarray(b);
  int i = 0, j = 0, cnt = 0;
  while (n - i >= m - j) {
    if (a[i] == b[j]) i++, j++;
    if (j == m) cnt++, j = lps[j - 1];
    else if (i < n && a[i] != b[j]) {
      if (j) j = lps[j - 1];
      else i++;
  } }
  return cnt;
}
```

5.3 Manacher

```cpp
void Manacher() {
  vector<int> d1(n);
```

```cpp
  // 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;
} }
```

5.4 String Match FFT

```cpp
/*find occurrences of t in s where '?'s are
automatically matched with any
 * character res[i + m - 1] = sum_j=0 to m - 1_{s[i + j]
 * t[j] * (s[i + j] -
 * t[j]) */
vector<int> string_matching(string& s, string& t) {
  int n = s.size(), m = t.size();
  vector<int> s1(n), s2(n), s3(n);
  for (int i = 0; i < n; i++)
    s1[i] = s[i] == '?' ? 0 : s[i] - 'a' + 1; /*assign
      any non zero number for non '?'s */
  for (int i = 0; i < n; i++) s2[i] = s1[i] * s1[i];
  for (int i = 0; i < n; i++) s3[i] = s1[i] * s2[i];
  vector<int> t1(m), t2(m), t3(m);
  for (int i = 0; i < m; i++) t1[i] = t[i] == '?' ? 0 :
    t[i] - 'a' + 1;
  for (int i = 0; i < m; i++) t2[i] = t1[i] * t1[i];
  for (int i = 0; i < m; i++) t3[i] = t1[i] * t2[i];
  reverse(t1.begin(), t1.end());
  reverse(t2.begin(), t2.end());
  reverse(t3.begin(), t3.end());
  vector<int> s1t3 = multiply(s1, t3);
  vector<int> s2t2 = multiply(s2, t2);
  vector<int> s3t1 = multiply(s3, t1);
  vector<int> res(n);
```

```cpp
  for (int i = 0; i < n; i++) res[i] = s1t3[i] - s2t2[i]
    * 2 + s3t1[i];
  vector<int> oc;
  for (int i = m - 1; i < n; i++)
    if (res[i] == 0) oc.push_back(i - m + 1);
  return oc;
}
```

5.5 Z

```cpp
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;
}
```

5.6 double hashing

```cpp
// Some Primes : 1000000007, 1000000009, 1000000861,
1000099999(< 2 ^ 30)1088888881, 1111211111, 1500000001,
1481481481(< 2 ^ 31)
namespace Hashing {
#define ff first
#define ss second
const PLL M = {1e9 + 7, 1e9 + 9};
const LL base = 1259; const int N = 1e6 + 7;
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}; }
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 << ")";
}
```

```cpp
PLL pb[N];
void hashPre() { pb[0] = {1, 1}; for (int i = 1; i < N;
i++) pb[i] = (pb[i - 1] * base) % M; }
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;
}
PLL substringHash(const vector<PLL>& hashlist, int l,
int r) {
  return (hashlist[r] + (M - hashlist[l - 1]) * pb[r - l
    + 1]) % M;
}
PLL Hash(string s) { PLL ans = {0, 0}; for (int i = 0; i
 < s.size(); i++) ans = (ans * base + s[i]) % M; return
ans; }
PLL append(PLL cur, char c) { return (cur * base + c) %
M; }
PLL prepend(PLL cur, int k, char c) { return (pb[k] * c
+ cur) % M; } ///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; } ///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 inverse(PLL a) {
  if (M.ss == 1) return bigmod(a, M.ff - 2);
  return bigmod(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; } ///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;
```

```cpp
  return ans % M;
} }
```

5.7   suffix array

```cpp
const int N = 3e5 + 9;
const int LG = 18;
void induced_sort(const vector<int> &vec, int val_range,
 vector<int> &SA, const vector<bool> &sl, const vector<
int> &lms_idx) {
  vector<int> l(val_range, 0), r(val_range, 0);
  for (int c : vec) {
    if (c + 1 < val_range) ++l[c + 1];
    ++r[c];
  }
  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 >= 1 && 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 >= 1; --k, i
= SA[k])
    if (i >= 1 && !sl[i - 1])
      SA[--r[vec[i - 1]]] = i - 1;
}
vector<int> SA_IS(const vector<int> &vec, int val_range)
 {
  const int n = vec.size();
  vector<int> SA(n), lms_idx;
  vector<bool> sl(n);
  sl[n - 1] = false;
  for (int i = n - 2; i >= 0; --i) {
    sl[i] = (vec[i] > vec[i + 1] || (vec[i] == vec[i +
    1] && sl[i + 1]));
    if (sl[i] && !sl[i + 1]) lms_idx.push_back(i + 1);
  }
  reverse(lms_idx.begin(), lms_idx.end());
  induced_sort(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 (!sl[SA[i]] && SA[i] >= 1 && sl[SA[i] - 1])
```

```cpp
      new_lms_idx[k++] = SA[i];
  int cur = 0;
  SA[n - 1] = cur;
  for (size_t 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 = false;
    for (int a = i + 1, b = j + 1;; ++a, ++b) {
      if (vec[a] != vec[b]) {
        flag = true; break;
      }
      if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1]))
       {
        flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b
         - 1])); break;
      }
    }
    SA[j] = (flag ? ++cur : cur);
  }
  for (size_t i = 0; i < lms_idx.size(); ++i)
    lms_vec[i] = SA[lms_idx[i]];
  if (cur + 1 < (int)lms_idx.size()) {
    auto lms_SA = SA_IS(lms_vec, cur + 1);
    for (size_t i = 0; i < lms_idx.size(); ++i)
      new_lms_idx[i] = lms_idx[lms_SA[i]];
  }
  induced_sort(vec, val_range, SA, sl, new_lms_idx);
  return SA;
}
vector<int> suffix_array(const string &s, const int LIM
= 128) {
  vector<int> vec(s.size() + 1);
  copy(begin(s), end(s), begin(vec));
  vec.back() = '$';
  auto ret = SA_IS(vec, LIM);
  ret.erase(ret.begin());
  return ret;
}
struct SuffixArray {
  int n; string s;
  vector<int> sa, rank, lcp;
  vector<vector<int>> t;
```

```cpp
vector<int> lg;
SuffixArray() {}
SuffixArray(string _s) {
  n = _s.size();
  s = _s;
  sa = suffix_array(s);
  rank.resize(n);
  for (int i = 0; i < n; i++) rank[sa[i]] = i;
  costruct_lcp();
  prec();
  build();
}
void costruct_lcp() {
  int k = 0;
  lcp.resize(n - 1, 0);
  for (int i = 0; i < n; i++) {
    if (rank[i] == n - 1) {
      k = 0; continue;
    }
    int j = sa[rank[i] + 1];
    while (i + k < n && j + k < n && s[i + k] == s[j +
    k]) k++;
    lcp[rank[i]] = k;
    if (k) k--;
} }
void prec() {
  lg.resize(n, 0);
  for (int i = 2; i < n; i++) lg[i] = lg[i / 2] + 1;
}
void build() {
  int sz = n - 1;
  t.resize(sz);
  for (int i = 0; i < sz; i++) {
    t[i].resize(LG); t[i][0] = lcp[i];
  }
  for (int k = 1; k < LG; ++k) {
    for (int i = 0; i + (1 << k) - 1 < sz; ++i) {
      t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][
      k - 1]);
} } }
int query(int l, int r) { // minimum of lcp[l], ...,
lcp[r]
  int k = lg[r - l + 1];
  return min(t[l][k], t[r - (1 << k) + 1][k]);
}
```

```cpp
int get_lcp(int i, int j) { // lcp of suffix starting
from i and j
  if (i == j) return n - i;
  int l = rank[i], r = rank[j];
  if (l > r) swap(l, r);
  return query(l, r - 1);
}
int lower_bound(string &t) {
  int l = 0, r = n - 1, k = t.size(), ans = n;
  while (l <= r) {
    int mid = l + r >> 1;
    if (s.substr(sa[mid], min(n - sa[mid], k)) >= t)
    ans = mid, r = mid - 1;
    else l = mid + 1;
  } return ans;
}
int upper_bound(string &t) {
  int l = 0, r = n - 1, k = t.size(), ans = n;
  while (l <= r) {
    int mid = l + r >> 1;
    if (s.substr(sa[mid], min(n - sa[mid], k)) > t) ans
    = mid, r = mid - 1;
    else l = mid + 1;
  } return ans;
}
// occurrences of s[p, ..., p + len - 1]
pair<int, int> find_occurrence(int p, int len) {
  p = rank[p];
  pair<int, int> ans = {p, p};
  int l = 0, r = p - 1;
  while (l <= r) {
    int mid = l + r >> 1;
    if (query(mid, p - 1) >= len) ans.first = mid, r =
    mid - 1;
    else l = mid + 1;
  }
  l = p + 1, r = n - 1;
  while (l <= r) {
    int mid = l + r >> 1;
    if (query(p, mid - 1) >= len) ans.second = mid, l =
    mid + 1;
    else r = mid - 1;
  } return ans;
} };
```

6   DP

6.1   CHT

```cpp
struct Line {
  mutable LL m, c, p;
  bool operator<(const Line& o) const { return m < o.m;
  }
  bool operator<(LL x) const { return p < x; }
};
/* this calculates maximum value of m * x + c over all
lines */
/* to get minimum value use m = -m , c = -c , query(x) =
 -query(x) */
struct LineContainer : multiset<Line, less<>> {
  /* (for doubles, use inf = 1/.0, div(a,b) = a/b) */
  static const LL inf = LLONG_MAX;
  LL div(LL a, LL b) { /* floored division */
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->m == y->m)
      x->p = x->c > y->c ? inf : -inf;
    else
      x->p = div(y->c - x->c, x->m - y->m);
    return x->p >= y->p;
  }
  void add(LL m, LL c) {
    auto z = insert({m, c, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y =
    erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect
    (x, erase(y));
  }
  LL query(LL x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.m * x + l.c;
  }
};
```

6.2   CatalanDp

```cpp
// comb formula: ((2n)Cn)-((2n)C(n-1)) = (1/(n+1))*((2n)
Cn)
void genCatalan(int n) {
  catalan[0] = catalan[1] = 1;
```

```cpp
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;
    } } }
}
```

6.3 DearrangementDP

```cpp
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;
}
```

6.4 Li Chao Tree

```cpp
using ll = long long;
const ll inf = 2e18;
struct Line {
  ll m, c;
  ll eval(ll x) {
    return m * x + c;
  }
};
struct node {
  Line line;
  node* left = nullptr;
  node* right = nullptr;
  node(Line line) : line(line) {}
  void add_segment(Line nw, int l, int r, int L, int R)
  {
    if (l > r || r < L || l > R) return;
    int m = (l + 1 == r ? l : (l + r) / 2);
    if (l >= L and r <= R) {
      bool lef = nw.eval(l) < line.eval(l);
      bool mid = nw.eval(m) < line.eval(m);
      if (mid) swap(line, nw);
      if (l == r) return;
      if (lef != mid) {
        if (left == nullptr) left = new node(nw);
        else left -> add_segment(nw, l, m, L, R);
      }
      else {
```

```cpp
        if (right == nullptr) right = new node(nw);
        else right -> add_segment(nw, m + 1, r, L, R);
      }
      return;
    }
    if (max(l, L) <= min(m, R)) {
      if (left == nullptr) left = new node({0, inf});
      left -> add_segment(nw, l, m, L, R);
    }
    if (max(m + 1, L) <= min(r, R)) {
      if (right == nullptr) right = new node ({0, inf});
      right -> add_segment(nw, m + 1, r, L, R);
    }
  }
  ll query_segment(ll x, int l, int r, int L, int R) {
    if (l > r || r < L || l > R) return inf;
    int m = (l + 1 == r ? l : (l + r) / 2);
    if (l >= L and r <= R) {
      ll ans = line.eval(x);
      if (l < r) {
        if (x <= m && left != nullptr) ans = min(ans,
        left -> query_segment(x, l, m, L, R));
        if (x > m && right != nullptr) ans = min(ans,
        right -> query_segment(x, m + 1, r, L, R));
      }
      return ans;
    }
    ll ans = inf;
    if (max(l, L) <= min(m, R)) {
      if (left == nullptr) left = new node({0, inf});
      ans = min(ans, left -> query_segment(x, l, m, L, R)
      );
    }
    if (max(m + 1, L) <= min(r, R)) {
      if (right == nullptr) right = new node ({0, inf});
      ans = min(ans, right -> query_segment(x, m + 1, r,
      L, R));
    }
    return ans;
  }
};
struct LiChaoTree {
  int L, R;
  node* root;
  LiChaoTree() : L(numeric_limits<int>::min() / 2), R(
  numeric_limits<int>::max() / 2), root(nullptr) {}
```

```cpp
  LiChaoTree(int L, int R) : L(L), R(R) {
    root = new node({0, inf});
  }
  void add_line(Line line) {
    root -> add_segment(line, L, R, L, R);
  }
  // y = mx + b: x in [l, r]
  void add_segment(Line line, int l, int r) {
    root -> add_segment(line, L, R, l, r);
  }
  ll query(ll x) {
    return root -> query_segment(x, L, R, L, R);
  }
  ll query_segment(ll x, int l, int r) {
    return root -> query_segment(x, l, r, L, R);
  }
};
```

6.5 SOS

```cpp
for (int mask = 0; mask < (1 << N); ++mask) {
  dp[mask][-1] = A[mask];
  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];
}
// good for inc-exc
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)];
```

6.6 grundy

```cpp
/* 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 */
```

```cpp
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);
}
```

7 Geometry

7.1 2D everything

```cpp
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);
}
int sign(double x) { return (x > eps) - (x < -eps); }

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 DT orientation (const point &a, const point &b,
  const point &c) { return (b - a) & (c - a); }
  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);
  }
  friend double Angle(const point& A, const point& B,
  const point& C) {
    point BA = A - B;
    point BC = C - B;
    double dot = BA ^ BC;
    double magBA = sqrt(~BA);
    double magBC = sqrt(~BC);
    double cosTheta = dot / (magBA * magBC);
    cosTheta = std::max(-1.0, std::min(1.0, cosTheta));
    return acos(cosTheta);
  }
  point Perp(const point& a) { return point(-a.y, a.x);
  }
  point Conj(const point& a) { return point(a.x, -a.y);
  }
};

/* given a pivot point and an initial direction, sorts
by Angle with the given direction */
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))
```

```cpp
      : O(O), dir(dir) {}
  bool operator()(point<DT> p, point<DT> q) {
    return make_tuple(half(p), 0) < make_tuple(half(q),
    (p & q));
  }
};

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>&
```

```cpp
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++) {
```

```cpp
    for (; m > 1 && orientation (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 && orientation (hull[m - 2], hull[m -
1], PT[i]) < 0; m--);
    hull[m++] = PT[i];
  }
  if (n > 1) m--;
  hull.resize(m);
  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; }

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. */ }

// -1 if strictly inside, 0 if on the polygon, 1 if
strictly outside
// it must be strictly convex, otherwise make it
strictly convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { //
O(log n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b = orientation(
p[0], p[n - 1], x);
```

```cpp
        if (a < 0 || b > 0) return 1;
        int l = 1, r = n - 1;
        while (l + 1 < r) {
            int mid = l + r >> 1;
            if (orientation(p[0], p[mid], x) >= 0) l = mid;
            else r = mid;
        }
        int k = orientation(p[l], p[r], x);
        if (k <= 0) return -k;
        if (l == 1 && a == 0) return 0;
        if (r == n - 1 && b == 0) return 0;
        return -1;
}

// -1 --> outside, 0 --> boundary, 1 --> inside
int pointInPolygon(vector<point> &pol, point z) { int n
= pol.size(), winding = 0; for(int i = 0; i < n; i++) {
point p1 = pol[i], p2 = pol[(i + 1) % n]; if(
isPointOnSegment(p1, p2, z)) return 0; int ori = sign(
cross(p1, p2, z)); if(p1.y <= z.y && p2.y > z.y && ori >
0) winding++; if(p1.y > z.y && p2.y <= z.y && ori < 0)
winding--; } return winding != 0 ? 1 : -1; }

} /* 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; }
```

7.2 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)
{} Point operator&(const Point& rhs) const { /* cross
product */ return Point(y * rhs.z - z * rhs.y, z * rhs.x
 - x * rhs.z, x * rhs.y - y * rhs.x); } DT operator^(
 const Point& rhs) const { /* dot product */ return x *
rhs.x + y * rhs.y + z * rhs.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); } };
```

7.3 MinDisSquares

```cpp
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;
```

```cpp
  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;
}
```

7.4 Picks Theorem

```cpp
struct Point {
  long long x, y;
  Point() {}
  Point(long long x, long long y) : x(x), y(y) {}
}; /// twice the area of polygon
long long double_area(Point poly[], int n) {
```

```
  long long res = 0;
  for (int i = 0, j = n - 1; i < n; j = i++) {
    res += ((poly[j].x + poly[i].x) * (poly[j].y - poly[
    i].y));
  }
  return abs(res);
} /// number of lattice points strictly on polygon
border
long long on_border(Point poly[], int n) {
  long long res = 0;
  for (int i = 0, j = n - 1; i < n; j = i++) {
    res += __gcd(abs(poly[i].x - poly[j].x), abs(poly[i
    ].y - poly[j].y));
  }
  return res;
} /// number of lattice points strictly inside polygon
long long interior(Point poly[], int n) {
  long long res = 2 + double_area(poly, n) - on_border(
  poly, n);
  return res / 2;
}
```

7.5   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;
```

```cpp
    if (dcmp(dot(poly[c] - poly[(c + 1) % n], u)) >= 0
&&
        dcmp(dot(poly[c] - poly[(c - 1 + n) % n], u)) >=
        0) {
      return c;
    }
    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);
```

```cpp
  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;
```

```cpp
}
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 Misc

8.1 All Macros

```cpp
//#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>;
struct custom_hash { static uint64_t splitmix64 (
uint64_t x) { x += 0x9e3779b97f4a7c15; x = (x ^ (x >>
30)) * 0xbf58476d1ce4e5b9; x = (x ^ (x >> 27)) * 0
x94d049bb133111eb; 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;

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);int getWeekday (int day, int month,
int year) { if (month <= 2) { month += 12; year -= 1; }
int f = (day + (13 * (month + 1)) / 5 + year + year / 4
- year / 100 + year / 400) % 7; return f;}
```

8.2 StressTest

```bash
#!/bin/bash
# Call as sh stress.sh ITERATIONS
g++ brute.cpp -o brute # brute solution
g++ cand.cpp -o cand # cand solution
g++ gen.cpp -o gen # test case gen
> all.txt
for i in $(seq 1 "$1") ; do
    echo "Attempt $i/$1"
    ./gen > in.txt
    echo "Attempt $i/$1" >> all.txt
    cat < in.txt >> all.txt
    ./brute < in.txt > out1.txt
    ./cand < in.txt > out2.txt
    diff -y out1.txt out2.txt > diff.txt
    if [ $? -ne 0 ] ; then
        echo -e "\nTest case:"
        cat in.txt
        echo -e "\nOutputs:"
        cat diff.txt
        break
    fi
done
files=("in.txt" "out1.txt" "out2.txt" "diff.txt" "cand"
"brute" "gen")
for file in "${files[@]}"; do
    rm "$file"
done
```

8.3 dbg

```cpp
void show(int x) {cerr << x;}
void show(long long x) {cerr << x;}
void show(double x) {cerr << x;}
void show(char x) {cerr << '\'' << x << '\'';}
void show(const string &x) {cerr << '\"' << x << '\"';}
void show(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void show(pair<T, V> x) { cerr << '{'; show(x.first);
cerr << ", "; show(x.second); cerr << '}'; }
template<typename T>
void show(T x) {int f = 0; cerr << "{"; for (auto &i: x)
 cerr << (f++ ? ", " : ""), show(i); cerr << "}";}

void debug_out(string s) {
    cerr << '\n';
}
template <typename T, typename... V>
void debug_out(string s, T t, V... v) {
    s.erase(remove(s.begin(), s.end(), ' '), s.end());
    cerr << "        "; // 8 spaces
    cerr << s.substr(0, s.find(','));
    s = s.substr(s.find(',') + 1);
```

```cpp
    cerr << " = ";
    show(t);
    cerr << endl;
    if(sizeof...(v)) debug_out(s, v...);
}

#define dbg(x...) cerr << "LINE: " << __LINE__ << endl;
debug_out(#x, x); cerr << endl;
```

8.4 gen

```cpp
mt19937 rng(random_device{}());
LL randomInt(LL low, LL high) {
  uniform_int_distribution<LL> dist(low, high);
  return dist(rng);
}
vector<pair<int, int>> generate_random_tree(int n) {
  vector<int> prufer(n - 2);
  for (int i = 0; i < n - 2; ++i) prufer[i] = randomInt
  (1, n);
  vector<int> degree(n + 1, 1);
  for (int x : prufer) degree[x]++;
  priority_queue<int, vector<int>, greater<int>> pq;
  for (int i = 1; i <= n; ++i)
    if (degree[i] == 1) pq.push(i);
  vector<pair<int, int>> edges;
  for (int x : prufer) {
    int leaf = pq.top();
    pq.pop();
    edges.emplace_back(leaf, x);
    degree[leaf]--, degree[x]--;
    if (degree[x] == 1) pq.push(x);
  }
  int u = pq.top(); pq.pop();
  int v = pq.top(); pq.pop();
  edges.emplace_back(u, v);
  return edges;
}
vector<int> permutation(int n) {
  vector<int> p(n); iota(p.begin(), p.end(), 1);
  shuffle(p.begin(), p.end(), rng);
  return p;
}
```

9 Equations and Formulas

9.1 Catalan Numbers

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

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.

9.2 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

$$\left[_n \, {}^n_- {}_k\right] = \sum_{0 \le i_1 < i_2 < i_k < n} i_1 i_2 .... i_k.$$

9.3 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 $1, 2, ..., n$. Define the reduced Stirling numbers of the second kind, denoted $S^d(n, k)$, to be the number of ways to partition the integers $1, 2, ..., n$ 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,
$S^d(n, k) = S(n - d + 1, k - d + 1), n \ge k \ge d$

9.4 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,

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

9.5 Different Math Formulas

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

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

9.6 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 $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
$\gcd(a, \text{lcm}(b, c)) = \text{lcm}(\gcd(a, b), \gcd(a, c))$.
$\text{lcm}(a, \gcd(b, c)) = \gcd(\text{lcm}(a, b), \text{lcm}(a, c))$.
For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$
$$\gcd(a, b) = \sum_{k|a \text{ and } k|b} \phi(k)$$

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

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

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

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

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

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

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

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

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

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