KUET_Potol Team Notebook

Mohammad Abu Daud Sharif,
Dipra Datta, Khadimul Islam Mahi

Contents

1 Combinatorics

1.1 nCr [22 lines]

```
const int MOD=1e9+7;
const int N=4e5+2;
int fact[N], ifact[N], reciprocal[N];

struct combinit {
  combinit() {
    reciprocal[1] = 1;
    for (int i = 2; i < N; ++i) {
      reciprocal[i] = (MOD - MOD / i) * (long
          long)reciprocal[MOD % i] % MOD;
    }
    fact[0] = ifact[0] = 1;
    for (int i = 1; i < N; ++i) {
      fact[i] = (long long)fact[i - 1] * i %
          MOD;
      ifact[i] = (long long)ifact[i - 1] *
          reciprocal[i] % MOD;
    }
  }
} combinitX;

long long comb(long long n, long long m) {
  if (n < m || m < 0) return 0;
  return (long long)fact[n] * ifact[m] % MOD
      * ifact[n - m] % MOD;
}
```

2 Data Structure

2.1 2D Fenwick Tree [19 lines]

```
struct FenwickTree2D
{
    vector<vector<int>> tree;
    int n;
    void init(int size){n = size,tree.assign(n
        + 1, vector<int>(n + 1, 0));}
    void update(int x, int y, int delta)
    {
        for (int i = x; i <= n; i += (i & -i))
            for (int j = y; j <= n; j += (j &
                -j))tree[i][j] += delta;
    }
    int query(int x, int y)
    {
        int sum = 0;
        for (int i = x; i > 0; i -= (i & -i))
            for (int j = y; j > 0; j -= (j &
                -j))sum += tree[i][j];
        return sum;
    }
    int query(int x1, int y1, int x2, int
        y2){return query(x2, y2) - query(x1 -
        1, y2) - query(x2, y1 - 1) + query(x1
        - 1, y1 - 1);}
};
```

2.2 2D Prefix Sum [3 lines]

```
void build(){for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j)prefix[i][j]
        = arr[i][j] + prefix[i - 1][j] +
        prefix[i][j - 1] - prefix[i - 1][j -
        1];}
int query(int x1, int y1, int x2, int
    y2){return prefix[x2][y2] - prefix[x1 -
    1][y2] - prefix[x2][y1 - 1] + prefix[x1 -
    1][y1 - 1];}
```

2.3 Centroid Decomposition [133 lines]

```
#include <bits/stdc++.h>
using namespace std;

struct CentroidDecomposition
{
    int n;
    int INF = INT_MAX;
    vector<vector<int>> adj;
    vector<bool> centroidMarked;
    vector<int> subSize;
    vector<int> parentCentroid;
    vector<vector<pair<int, int>>>
        centroidPath;
    vector<multiset<int>> bestSet;
    CentroidDecomposition(int _n) : n(_n)
    {
        adj.assign(n + 1, {});
        centroidMarked.assign(n + 1, false);
        subSize.assign(n + 1, 0);
        parentCentroid.assign(n + 1, 0);
        centroidPath.assign(n + 1, {});
        bestSet.assign(n + 1, {});
    }
    void addEdge(int u, int v)
    {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void computeSubSizeDFS(int u, int p)
    {
        subSize[u] = 1;
        for (int v : adj[u])
        {
            if (v == p || centroidMarked[v])
                continue;
            computeSubSizeDFS(v, u);
            subSize[u] += subSize[v];
        }
    }
    int findCentroidDFS(int u, int p, int
        totSize)
    {
        for (int v : adj[u])
        {
            if (v == p || centroidMarked[v])
                continue;
            if (subSize[v] > totSize / 2)
            {
                return findCentroidDFS(v, u,
                    totSize);
            }
        }
        return u;
    }
    void addCentroidDistances(int c)
    {
        queue<pair<int, int>> q;
        vector<char> visited(n + 1, 0);
        q.push({c, 0});
        visited[c] = 1;
```

```
    centroidPath[c].push_back({c, 0});
    while (!q.empty())
    {
        auto [u, d] = q.front();
        q.pop();
        for (int v : adj[u])
        {
            if (centroidMarked[v] ||
                visited[v])
                continue;
            visited[v] = 1;
            centroidPath[v].push_back({c,
                d + 1});
            q.push({v, d + 1});
        }
    }
}
void decompose(int entry, int pCent)
{
    computeSubSizeDFS(entry, 0);
    int totalSize = subSize[entry];

    int c = findCentroidDFS(entry, 0,
        totalSize);

    centroidMarked[c] = true;
    parentCentroid[c] = (pCent == 0 ? c :
        pCent);

    addCentroidDistances(c);

    for (int v : adj[c])
    {
        if (!centroidMarked[v])
        {
            decompose(v, c);
        }
    }
}
void build()
{
    decompose(1, 0);
}
void update(int v)
{
    for (auto &pr : centroidPath[v])
    {
        int c = pr.first;
        int d = pr.second;
        bestSet[c].insert(d);
    }
}
void remove(int v)
{
    for (auto &pr : centroidPath[v])
    {
        int c = pr.first;
        int d = pr.second;
        auto it = bestSet[c].find(d);
        if (it != bestSet[c].end())
        {
            bestSet[c].erase(it);
        }
    }
}
int query(int v)
{
    int res = INF;
    for (auto &pr : centroidPath[v])
    {
        int c = pr.first;
        int d = pr.second;
        if (!bestSet[c].empty())
        {
            res = min(res, d +
                *bestSet[c].begin());
        }
    }
    return res;
}
};
```

2.4 DSU with Rollback [33 lines]

```
struct DSU
{
    vector<int> parent, size;
    vector<pair<int, int>> history;
    int componentCount;
    DSU(int n) : parent(n + 1), size(n + 1,
        1), componentCount(n){iota(parent.
        begin(), parent.end(), 0);}
    int find(int v)
    {
        if (parent[v] != v)return
            find(parent[v]);
        return parent[v];
    }
    bool merge(int u, int v)
    {
        u = find(u),v = find(v);
        if (u == v){return false;}
        if (size[u] < size[v])swap(u, v);
        history.push_back({v, size[v]});
        history.push_back({u, size[u]});
        parent[v] = u,size[u] +=
            size[v],--componentCount;
        return true;
    }
    void rollback()
    {
        auto [u, oldSizeU] = history.back();
        history.pop_back();
        auto [v, oldSizeV] = history.back();
        history.pop_back();
        parent[v] = v,size[u] =
            oldSizeU,size[v] =
            oldSizeV,componentCount++;
    }
    int getComponentCount(){return
        componentCount;}
    bool same(int u, int v){return find(u) ==
        find(v);}
    int getSize(int v){return size[find(v)];}
};
```

2.5 Fenwick Tree [46 lines]

```
struct FenwickTree
{
    vector<int> tree;
    int n;
    FenwickTree(int n){this->n =
        n,tree.resize(n + 1, 0);}
    void update(int idx, int val)
    {
        while (idx <= n)tree[idx] += val,idx
            += idx & (-idx);
    }
    int query(int idx)
    {
        int sum = 0;
        while (idx > 0)sum += tree[idx],idx
            -= idx & (-idx);
        return sum;
    }
    int rangeQuery(int l, int r){ return
        query(r) - query(l - 1);}
    int inversion_count(vector<int> &arr)
    {
        int icount = 0;
        for (int i = n; i > 0; i--)
        {
            int smallerCount = query(arr[i] -
                1);
            icount += smallerCount;
            update(arr[i], 1);
        }
        return icount;
    }

    int kth(int k)
    {
        if (k <= 0 || query(n) < k) return -1;
        int idx = 0, mask = 1 << (31 -
            __builtin_clz(n));
        int acc = 0;
        while (mask)
        {
            int next = idx + mask;
            if (next <= n && acc + tree[next]
                < k)
            {
                idx = next;
                acc += tree[next];
            }
            mask >>= 1;
        }
        return idx + 1;
    }
};
```

2.6 HeavyLight Decomposition [72 lines]

```
struct HLD{
    vector<vector<int>> adj;
    vector<int> par, dep, heavy, head, pos,
        sub;
    SegTree segTree;
    int curPos;
    HLD(int n) : adj(n + 1),par(n + 1,
        -1),dep(n + 1, 0),heavy(n + 1,
        -1),head(n + 1),pos(n + 1), sub(n +
        1, 0),segTree(n),curPos(1) {}
    int dfs(int u)
    {
        sub[u] = 1;
        int mxsub = 0;
        for (int v : adj[u])
        {
            if (v == par[u])
                continue;
            par[v] = u;
            dep[v] = dep[u] + 1;
            int subtree = dfs(v);
            sub[u] += subtree;
            if (subtree > mxsub)
            {
                mxsub = subtree;
                heavy[u] = v;
            }
        }
        return sub[u];
    }
    void decompose(int u, int h)
    {
        head[u] = h;
        pos[u] = curPos++;
        if (heavy[u] != -1)
            decompose(heavy[u], h);
        for (int v : adj[u])
        {
            if (v != par[u] && v != heavy[u])
                decompose(v, v);
        }
    }
    void init(int root = 1)
    {
        dfs(root);
        decompose(root, root);
    }
    void updatePath(int u, int v, ll value,
        bool isEdge = false)
    {
        while (head[u] != head[v])
        {
            if (dep[head[u]] < dep[head[v]])
                swap(u, v);
            segTree.update(pos[head[u]],
                pos[u], value);
            u = par[head[u]];
        }
        if (dep[u] > dep[v])
            swap(u, v);
        segTree.update(pos[u] + isEdge,
            pos[v], value);
    }
    ll queryPath(int u, int v, bool isEdge =
        false)
    {
        ll result = 0;
        while (head[u] != head[v])
        {
            if (dep[head[u]] < dep[head[v]])
                swap(u, v);
            result = max(result,
                segTree.query(pos[head[u]],
                pos[u]));
            u = par[head[u]];
        }
        if (dep[u] > dep[v])
            swap(u, v);
        result = max(result,
            segTree.query(pos[u] + isEdge,
            pos[v]));
```

```
        return result;
    }
};
```

2.7 LCA [37 lines]

```
const int N=3e5+2;
vector<int>g[N];
int timer;
int stime[N];
int etime[N];
int pp[N][20];
void dfs(int node,int par)
{
    stime[node]=++timer;
    pp[node][0]=par;
    for(int i=1;i<20;i++)pp[node][i]=
        pp[pp[node][i-1]][i-1];
    for(auto it:g[node])
    {
        if(it==par)continue;
        dfs(it,node);
    }
    etime[node]=++timer;
}
int isansistor(int x,int y)
{
    if(x==0)return 1;
    return stime[x]<=stime[y]&&etime[x]>=
        etime[y];
}
int lca(int x,int y)
{
    if(stime[x]>stime[y])swap(x,y);
    if(isansistor(x,y))return x;
    for(int i=19;i>=0;i--)
    {
        int tem=pp[x][i];
        if(isansistor(tem,y)==0)
        {
            x=tem;
        }
    }
    return pp[x][0];
}
```

2.8 Lazy Segment Tree [76 lines]

```
using ll = long long;
struct Node {
  ll a;
  Node() : a(LLONG_MIN/4) {}
  Node(ll _a) : a(_a) {}
  friend Node merge(const Node &A, const Node
      &B) {
    return Node(max(A.a, B.a));
  }
};

struct SegmentTree {
  int n;
  int size;
  vector<Node> t;
  vector<ll> lazy;
  vector<bool> isLazy;

  SegmentTree(int _n) { init(_n); }

  void init(int _n) {
    n = _n;
    size = 1;
    while (size < max(1, n)) size <<= 1;
    t.assign(2 * size, Node());
    lazy.assign(2 * size, 0);
    isLazy.assign(2 * size, 0);
  }

  void build(const vector<ll> &vals) {
    for (int i = 1; i <= n; ++i) t[size + i -
        1] = Node(vals[i]);
    for (int i = n + 1; i <= size; ++i) t[size
        + i - 1] = Node();
    for (int nd = size - 1; nd >= 1; --nd)
        pull(nd);
  }

  inline void pull(int nd) {
    t[nd] = merge(t[nd << 1], t[nd << 1 | 1]);
  }
  inline void apply_node(int nd, int st, int
      en, ll val) {
    t[nd].a += val;
    lazy[nd] += val;
    isLazy[nd] = 1;
  }

  inline void push(int nd, int st, int en) {
    if (!isLazy[nd] || nd >= size) return;
    int mid = (st + en) >> 1;
    apply_node(nd << 1, st, mid, lazy[nd]);
    apply_node(nd << 1 | 1, mid+1, en,
        lazy[nd]);
    isLazy[nd] = 0;
    lazy[nd] = 0;
  }
  void update(int l, int r, ll val) {
      update(1, 1, size, l, r, val); }
  void update(int nd, int st, int en, int l,
      int r, ll val) {
    if (l > en || r < st) return;
    if (l <= st && en <= r) {
      apply_node(nd, st, en, val);
      return;
    }
    push(nd, st, en);
    int mid = (st + en) >> 1;
    update(nd << 1, st, mid, l, r, val);
    update(nd << 1 | 1, mid + 1, en, l, r,
        val);
    pull(nd);
  }

  Node query(int l, int r) { return query(1,
      1, size, l, r); }
  Node query(int nd, int st, int en, int l,
      int r) {
    if (l > en || r < st) return Node();
    if (l <= st && en <= r) return t[nd];
    push(nd, st, en);
    int mid = (st + en) >> 1;
    Node L = query(nd << 1, st, mid, l, r);
    Node R = query(nd << 1 | 1, mid + 1, en,
        l, r);
    return merge(L, R);
  }
};
```

2.9 Mo's Algorithm [29 lines]

```
const int N = 2e5 + 5;
const int Q = 2e5 + 5;
const int SZ = sqrt(N) + 1;
struct qry {
  int l, r, id, blk;
  bool operator<(const qry& p) const {
    if(blk == p.blk) return blk%2?r>p.r:r<p.r;
    return blk<p.blk;
  }
};
qry query[Q];
ll ans[Q];
void add(int id) {}
void remove(int id) {}
ll get() {}
int n, q;
void MO() {
  sort(query, query + q);
  int cur_l = 0, cur_r = -1;
  for (int i = 0; i < q; i++) {
    qry q = query[i];
    while (cur_l > q.l) add(--cur_l);
    while (cur_r < q.r) add(++cur_r);
    while (cur_l < q.l) remove(cur_l++);
    while (cur_r > q.r) remove(cur_r--);
    ans[q.id] = get();
  }
}
/* 0 indexed. */
```

2.10 Persistent Segment Tree [142 lines]

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

struct Persistent
{
    struct Node
    {
        long long sum, min_val, max_val;
        Node *left, *right;

        Node() : sum(0), min_val(LLONG_MAX),
            max_val(LLONG_MIN),
            left(nullptr), right(nullptr) {}
    };

    Node *root;
    int n;

    Persistent(int n)
    {
        this->n = n;
        root = build(1, n);
    }

    Node *build(int st, int ed)
    {
        Node *node = new Node();
        if (st == ed)
        {
            return node;
        }
        int mid = (st + ed) / 2;
        node->left = build(st, mid);
        node->right = build(mid + 1, ed);
        return node;
    }

    Node merge(const Node &left, const Node
        &right)
    {
        Node result;
        result.sum = left.sum + right.sum;
        result.min_val =
            std::min(left.min_val,
            right.min_val);
        result.max_val =
            std::max(left.max_val,
            right.max_val);
        return result;
    }

    Node *update(Node *node, int st, int ed,
        int idx, long long val)
    {
        Node *nd = new Node(*node);
        if (st == ed)
        {
            nd->sum = val;
            nd->min_val = val;
            nd->max_val = val;
            return nd;
        }
        int mid = (st + ed) / 2;
        if (idx <= mid)
        {
            nd->left = update(node->left, st,
                mid, idx, val);
        }
        else
        {
            nd->right = update(node->right,
                mid + 1, ed, idx, val);
        }
        Node left_result = nd->left ?
            *(nd->left) : Node();
        Node right_result = nd->right ?
            *(nd->right) : Node();
        Node merged_result =
            merge(left_result, right_result);
        nd->sum = merged_result.sum;
        nd->min_val = merged_result.min_val;
        nd->max_val = merged_result.max_val;
        return nd;
    }

    Node query(Node *node, int st, int ed, int
        l, int r)
    {
        if (!node || st > r || ed < l)
        {
            return Node();
        }
        if (st >= l && ed <= r)
        {
            return *node;
```

```
        }
        int mid = (st + ed) / 2;
        Node left_result = query(node->left,
            st, mid, l, r);
        Node right_result =
            query(node->right, mid + 1, ed,
            l, r);
        return merge(left_result,
            right_result);
    }

    Node *updateIndex(Node *root, int idx,
        long long val)
    {
        return update(root, 1, n, idx, val);
    }

    Node queryRange(Node *root, int l, int r)
    {
        return query(root, 1, n, l, r);
    }
};

void solve()
{
    int n, q;
    cin >> n >> q;
    vector<ll> v(n + 1);
    for (int i = 1; i <= n; i++)
    {
        cin >> v[i];
    }
    Persistent obj(n);
    Persistent::Node *root = obj.root;
    for (int i = 1; i <= n; i++)
    {
        root = obj.updateIndex(root, i, v[i]);
    }
    vector<Persistent::Node *> version;
    version.push_back(root);
    while (q--)
    {
        int op;
        cin >> op;
        if (op == 1)
        {
            ll k, a, x;
            cin >> k >> a >> x;
            Persistent::Node *newRoot =
                obj.updateIndex(version[k -
                1], a, x);
            version[k - 1] = newRoot;
        }
        else if (op == 2)
        {
            ll k, a, b;
            cin >> k >> a >> b;
            cout << obj.queryRange(version[k
                - 1], a, b).sum << endl;
        }
        else
        {
            int k;
            cin >> k;
            version.push_back(version[k - 1]);
        }
    }
}
```

2.11 Sparse Table [47 lines]

```
struct sparse_table
{
    // for converting 1 based indexing
        initialize n=size+1,s[0]=dummy;
    ll mxn;
    ll k;
    vector<vector<ll>> table1, table2;
    vector<ll> logs;
    sparse_table(ll n)
    {
        mxn = n;
        logs.resize(n + 1, 0);
        for (int i = 2; i <= mxn; i++)
        {
            logs[i] = logs[i >> 1] + 1;
        }
        k = logs[mxn] + 1;
        table1.resize(mxn + 1, vector<ll>(k));
        table2.resize(mxn + 1, vector<ll>(k));
    }
    void create_table(vector<ll> &v)
    {
        for (int i = 0; i < mxn; i++)
        {
            table1[i][0] = table2[i][0] =
                v[i];
        }
        for (ll i = 1; i < k; i++)
        {
            for (ll j = 0; j + (1 << i) - 1 <
                mxn; j++)
            {
                table1[j][i] = min(table1[j][i
                    - 1], table1[j + (1 << (i
                    - 1))][i - 1]);
                table2[j][i] = max(table2[j][i
                    - 1], table2[j + (1 << (i
                    - 1))][i - 1]);
            }
        }
    }
    ll query1(ll x, ll y)
    {
        ll gap = y - x + 1;
        ll lg = logs[gap];
        return min(table1[x][lg], table1[y -
            (1 << lg) + 1][lg]);
    }
    ll query2(ll x, ll y)
    {
        ll gap = y - x + 1;
        ll lg = logs[gap];
        return max(table2[x][lg], table2[y -
            (1 << lg) + 1][lg]);
    }
};
```

2.12 TrieString [93 lines]

```
struct TNode
{
    TNode *child[26];
    bool isTerm;
    int cnt;
    TNode()
    {
        isTerm = false;
        cnt = 0;
        for (int i = 0; i < 26; i++)
        {
            child[i] = nullptr;
        }
    }
};

struct Trie
{
    TNode *root;
    Trie()
    {
        root = new TNode();
    }
    void insert(const string &s)
    {
        TNode *rt = root;
        for (char ch : s)
        {
            int idx = ch - 'a';
            if (rt->child[idx] == nullptr)
            {
                rt->child[idx] = new TNode();
            }
            rt = rt->child[idx];
            rt->cnt++;
        }
        rt->isTerm = true;
    }
    bool search(const string &s)
    {
        TNode *rt = root;
        for (int i = 0; i < s.size(); i++)
        {
            int idx = s[i] - 'a';
            if (rt->child[idx] == nullptr)
            {
                return false;
            }
            rt = rt->child[idx];
        }
        return rt->isTerm;
    }

    bool deleteHelper(TNode *rt, const string
        &s, int pos)
    {
        if (pos == s.length())
        {
            if (!rt->isTerm) return false;
            rt->isTerm = false;
            return (rt->cnt == 0);
        }
        int idx = s[pos] - 'a';
        if (rt->child[idx] == nullptr) return
            false;
        bool delch =
            deleteHelper(rt->child[idx], s,
            pos + 1);
        rt->child[idx]->cnt--;
        if (delch)
        {
            delete rt->child[idx];
            rt->child[idx] = nullptr;
        }
        return (rt->child[idx] == nullptr) &&
            !rt->isTerm;
    }
    void remove(const string &s)
    {
        deleteHelper(root, s, 0);
    }
    void clear(TNode *rt)
    {
        if (!rt)
        {
            return;
        }
        for (int i = 0; i < 26; i++)
        {
            clear(rt->child[i]);
        }
        delete rt;
    }
    ~Trie()
    {
        clear(root);
    }
};
```

2.13 TrieXOR [90 lines]

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

class Trie
{
public:
    static const int B = 32;
    struct Node
    {
        Node *nxt[2];
        int sz;
        Node() : sz(0)
        {
            nxt[0] = nxt[1] = nullptr;
        }
    };
    Node *root;
    Trie()
    {
        root = new Node();
    }
    // Insert a number into the Trie
    void insert(int val)
    {
        Node *cur = root;
        cur->sz++;
        for (int i = B - 1; i >= 0; --i)
        {
            int bit = (val >> i) & 1;
            if (!cur->nxt[bit]) cur->nxt[bit]
                = new Node();
            cur = cur->nxt[bit];
            cur->sz++;
        }
```

```
    }
    // Query: count numbers 'val' in the Trie
        such that (val XOR x) < k.
    int query(int x, int k)
    {
        Node *cur = root;
        int count = 0;
        for (int i = B - 1; i >= 0; --i)
        {
            if (!cur) break;
            int xBit = (x >> i) & 1, kBit = (k
                >> i) & 1;
            if (kBit == 1)
            {
                if (cur->nxt[xBit]) count +=
                    cur->nxt[xBit]->sz;
                cur = cur->nxt[1 - xBit];
            }
            else  cur = cur->nxt[xBit];
        }
        return count;
    }

    // Given x, return the maximum XOR
        achievable with any inserted number.
    int getMaxXor(int x)
    {
        Node *cur = root;
        int result = 0;
        for (int i = B - 1; i >= 0; --i)
        {
            int xBit = (x >> i) & 1;
            int desired = 1 - xBit;
            if (cur->nxt[desired])
            {
                result |= (1 << i);
                cur = cur->nxt[desired];
            }
            else cur = cur->nxt[xBit];
        }
        return result;
    }
    // Given x, return the minimum XOR
        achievable with any inserted number.
    int getMinXor(int x)
    {
        Node *cur = root;
        int result = 0;
        for (int i = B - 1; i >= 0; --i)
        {
            int xBit = (x >> i) & 1;
            if (cur->nxt[xBit]) cur =
                cur->nxt[xBit];
            else
            {
                result |= (1 << i);
                cur = cur->nxt[1 - xBit];
            }
        }
        return result;
    }
};
```

2.14 Wavelet Tree [101 lines]

```
using ll = long long;
struct WaveletTree
{
    int lo, hi;
    WaveletTree *L = nullptr, *R = nullptr;
    // b[i] = number of elements from
        prefix[0..i-1] that went to left
        child.
    // b.size() == n+1 where b[0] = 0.
    vector<int> b;
    vector<ll> pref;
    WaveletTree() = default;
    WaveletTree(const vector<int> &arr, int
        x, int y) : lo(x), hi(y), L(nullptr),
        R(nullptr)
    {
        int n = (int)arr.size();
        b.reserve(n + 1);
        pref.reserve(n + 1);
        b.push_back(0);
        pref.push_back(0);
        if (n == 0) return;
        if (lo == hi)
        {
            for (int v : arr)
            {
                b.push_back(b.back() + 1);
                pref.push_back(pref.back() +
                    v);
            }
            return;
        }
        int mid = (lo + hi) >> 1;
        vector<int> leftArr;
        leftArr.reserve(n);
        vector<int> rightArr;
        rightArr.reserve(n);
        for (int v : arr)
        {
            if (v <= mid)
            {
                leftArr.push_back(v);
                b.push_back(b.back() + 1);
            }
            else
            {
                rightArr.push_back(v);
                b.push_back(b.back());
            }
            pref.push_back(pref.back() + v);
        }
        if (!leftArr.empty())
            L = new WaveletTree(leftArr, lo,
                mid);
        if (!rightArr.empty())
            R = new WaveletTree(rightArr, mid
                + 1, hi);
    }
    ~WaveletTree()
    {
        delete L;
        delete R;
    }
    // kth smallest in [l, r], 1-based. returns
        -1 if invalid k.
    int kth(int l, int r, int k) const
    {
        if (l > r || k <= 0 || k > r - l + 1)
            return -1;
        if (lo == hi) return lo;
        int inLeft = b[r] - b[l - 1];
        if (k <= inLeft) return L->kth(b[l -
            1] + 1, b[r], k);
        else return R->kth(l - b[l - 1], r -
            b[r], k - inLeft);
    }
    // count of numbers <= k in [l, r]
    int countLTE(int l, int r, int k) const
    {
        if (l > r || k < lo) return 0;
        if (hi <= k) return r - l + 1;
        int lb = b[l - 1], rb = b[r];
        return (L ? L->countLTE(lb + 1, rb, k)
            : 0) + (R ? R->countLTE(l - lb, r
            - rb, k) : 0);
    }
    // count equal to value in [l, r]
    int countEqual(int l, int r, int value)
        const
    {
        if (l > r || value < lo || value > hi)
            return 0;
        if (lo == hi) return r - l + 1;
        int mid = (lo + hi) >> 1;
        int lb = b[l - 1], rb = b[r];
        if (value <= mid) return L ?
            L->countEqual(lb + 1, rb, value) :
            0;
        else return R ? R->countEqual(l - lb,
            r - rb, value) : 0;
    }
    // sum of values < k in [l, r]
    // Requires pref to be present (we store
        original prefix sums at construction)
    ll sumLess(int l, int r, int k) const
    {
        if (l > r || k <= lo) return 0;
        if (hi < k) return pref[r] - pref[l -
            1];
        int lb = b[l - 1], rb = b[r];
        ll leftSum = L ? L->sumLess(lb + 1,
            rb, k) : 0;
        ll rightSum = R ? R->sumLess(l - lb, r
            - rb, k) : 0;
        return leftSum + rightSum;
    }

    // utility: returns total number of
        elements stored at this node
    int size() const
    {
        return (int)pref.size() - 1;
    }
};
```

3 Dynamic Programming

3.1 Cartesian [42 lines]

```
// Build a max-heap Cartesian tree over
    `h[0..n-1]`.
// Returns the index of the root.
// Outputs left-child in L[], right-child in
    R[].
int buildCartesianTree(const vector<int> &h,
    vector<int> &L, vector<int> &R)
{
    int n = h.size();
    L.assign(n, -1);
    R.assign(n, -1);
    vector<int> parent(n, -1);
    stack<int> st;

    for (int i = 0; i < n; i++)
    {
        int last = -1;
        while (!st.empty() && h[st.top()] <
            h[i])
        {
            last = st.top();
            st.pop();
        }
        if (!st.empty())
        {
            parent[i] = st.top();
            R[st.top()] = i;
        }
        if (last != -1)
        {
            parent[last] = i;
            L[i] = last;
        }
        st.push(i);
    }
    int root = -1;
    for (int i = 0; i < n; i++)
    {
        if (parent[i] == -1)
        {
            root = i;
            break;
        }
    }
    return root;
}
```

3.2 Convex Hull Trick [99 lines]

```
********************Offline
    **********************
// dpcur[i] depends on dppre[1--n]
const ll M = 1e16 + 7;
const ll N = 1e5 + 3;
ll dis[N];
ll xcross(ll m1, ll c1, ll m2, ll c2, ll n)
{
    ld ans = ((ld)c1 - c2) / (m2 - m1);
    if (ans > n) ans = n + 1;
    return ceil(ans);
}
// main
int main()
{
    fastio int n, m, k;
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) dis[i] = M;
    for (ll i = 1; i <= k; i++)
    {
        vector<ll> str;
        str.push_back(1);
        str.push_back(2);
```

```
        for (ll i = 3; i <= n; i++)
        {
            while (str.size() > 1)
            {
                ll lst = str.back();
                ll slst = str[str.size() - 2];
                ll cur = xcross(-2 * lst,
                    dis[lst] + lst * lst, -2
                    * i, dis[i] + i * i, n);
                ll pre = xcross(-2 * lst,
                    dis[lst] + lst * lst, -2
                    * slst, dis[slst] + slst
                    * slst, n);
                if (cur <= pre)
                    str.pop_back();
                else break;
            }
            str.push_back(i);
        }
        ll pre = 1;
        ll diss[n + 1];
        for (ll i = 1; i < str.size(); i++)
        {
            ll lst = str[i];
            ll cur = xcross(-2 * lst, dis[lst]
                + lst * lst, -2 * i, dis[i] +
                i * i, n);
        }
        for (ll i = 1; i < str.size(); i++)
        {
            ll lst = str[i - 1];
            ll slst = str[i];
            ll cur = xcross(-2 * lst, dis[lst]
                + lst * lst, -2 * slst,
                dis[slst] + slst * slst, n);
            for (ll i = pre; i < cur; i++)
                diss[i] = (i - lst) * (i -
                lst) + dis[lst];
            pre = cur;
        }
        ll lst = str.back();
        for (ll i = pre; i <= n; i++) diss[i]
            = (i - lst) * (i - lst) +
            dis[lst];
        for (int i = 1; i <= n; i++) dis[i] =
            diss[i];
    }
    for (int i = 1; i <= n; i++) cout <<
        dis[i] << " ";
    cout << "\n";
}

// for(i>j) dp[i]=min(dp[j]+(i-j)^2+c);
    [1]=C0;
ll xcross(ll m1, ll c1, ll m2, ll c2, ll n)
{
    ld ans = ((ld)c1 - c2) / (m2 - m1);
    if (ans > n) ans = n + 1;
    return ceil(ans);
}
ll dp[N];
void CHT(ll n, ll C1, ll C)
{
    dp[1] = C1;
    vector<ll> str;
    str.push_back(1);
    int cur = 0;
    for (ll i = 2; i <= n; i++)
    {
        dp[i] = C + (i - str[cur]) * (i -
            str[cur]) + dp[str[cur]];
        while (cur < (int)str.size() - 1)
        {
            cur++;
            ll tem = C + (i - str[cur]) * (i
                - str[cur]) + dp[str[cur]];
            if (tem > dp[i])
            {
                cur--;
                break;
            }
            dp[i] = tem;
        }
        while (str.size() > 1)
        {
            ll lst = str.back();
            ll slst = str[str.size() - 2];
            ll cur = xcross(-2 * lst, dp[lst]
                + lst * lst, -2 * i, dp[i] + i
                * i, n);
            ll pre = xcross(-2 * lst, dp[lst]
                + lst * lst, -2 * slst,
                dp[slst] + slst * slst, n);
            if (cur <= pre) str.pop_back();
            else break;
        }
        if (cur >= str.size()) cur =
            str.size() - 1;
        str.push_back(i);
    }
}
```

3.3 Divide And Conquer Trick [41 lines]

```
int dcp(int st, int ed, int opt1, int opt2,
    int n)
{
    if (st > ed) return opt1;
    int mid = (st + ed) / 2;
    int opt = opt1;
    ll anss = inf;
    ll cost = ans(1, 1, n, min(mid, opt2) +
        1, mid, mid);
    for (int i = min(opt2, mid); i >= opt1;
        i--)
    {
        if (nxt[i] <= mid) cost = cost +
            nxt[i] - i;
        if (cost + tem[i - 1] < anss)
        {
            anss = cost + tem[i - 1];
            opt = i;
        }
    }
    tk[mid] = anss;
    if (st == ed) return opt;
    opt1 = dcp(st, mid - 1, opt1, opt, n);
    opt2 = dcp(mid + 1, ed, opt, opt2, n);
    return opt2;
}
//main
int main()
{
    fastio
    int n,k;
    cin>>n>>k;
    for(int i=1; i<=n; i++)cin>>a[i];
    gen(n);
    make(1,1,n);
    tk[0]=0;
    for(int i=1; i<=n; i++)tk[i]=0;
    for(int kk=1; kk<=k; kk++)
    {
        for(int i=0; i<=n; i++)tem[i]=tk[i];
        int opt=dcp(1,n,1,n,n);
    }
    cout<<tk[n]<<"\n";
}
```

3.4 SOS DP [88 lines]

```
const ll MLOG = 20;
const ll MAXN = (1<<MLOG);
ll dp[MAXN]; ll freq[MAXN];
void forward1(){ // adding element to all its
    super set
    for(ll bit = 0; bit < MLOG; bit++){
        for(ll i = 0; i < MAXN; i++){
            if(i&(1<<bit)){
                dp[i]+=dp[i^(1<<bit)]; //add
                    a[i] to a[j] if j&i = i
            }
        }
    }
}
void backward1(){
    for(ll bit = 0; bit < MLOG; bit++){
        for(ll i = MAXN-1; i >= 0; i--){
            if(i&(1<<bit)){
                dp[i]-=dp[i^(1<<bit)];
            }
        }
    }
}
void forward2(){ // add elements to its
    subsets
    for(ll bit = 0; bit < MLOG; bit++){
        for(ll i = MAXN-1; i >= 0; i--){
            if(i&(1<<bit)){
                dp[i^(1<<bit)]+=dp[i];
            }
        }
    }
}
void backward2(){
    for(ll bit = 0; bit < MLOG; bit++){
        for(ll i = 0; i < MAXN; i++){
            if(i&(1<<bit)){
                dp[i^(1<<bit)]-=dp[i];
            }
        }
    }
}

//F[i] = A[i]2 +(SUM(G[j]))2 where (i&j) = j
    and j < i
//G[i] = (SUM(F[j]2)) where (i&j) = j and j <=
    i
int main()
{
    fastio
    int n;
    cin>>n;
    ll a[n];
    for(int i=0;i<n;i++)cin>>a[i];
    int lg=log2(n)+2;
    ll f[n],g[n],gs[n][lg],f2s[n][lg];
    for(int i=0;i<n;i++)
    {
        f[i]=0;
        for(int j=0;j+1<lg;j++)
        {
            if(i&(1<<j))
            {
                f[i]=f[i]+gs[i^(1<<j)][j];
            }
        }
        f[i]%=M;
        f[i]=(f[i]*f[i]+a[i]*a[i])%M;
        f2s[i][0]=f[i]*f[i]%M;
        for(int j=0;j+1<lg;j++)
        {
            if(i&(1<<j))
            {
                f2s[i][j+1]=(f2s[i][j]+
                    f2s[(i^(1<<j))][j])%M;
            }else f2s[i][j+1]=f2s[i][j];
        }
        g[i]=f2s[i][lg-1];
        gs[i][0]=g[i];
        for(int j=0;j+1<lg;j++)
        {
            if(i&(1<<j))
            {
                gs[i][j+1]=(gs[i][j]+gs[i^(1<
                    <j)][j])%M;
            }else gs[i][j+1]=gs[i][j];
        }
    }
    ll sum=0;
    for(int i=0;i<n;i++)
    {
        sum=(sum+(f[i]*g[i]%M)*i)%M;
    }
    cout<<sum<<"\n";
}
```

4 Flow

4.1 Dinic [76 lines]

```
// O(V^2*E)
struct Dinic
{
    vector<vector<int>> adj, capacity;
    vector<int> level, ptr;
    int n;

    Dinic(int n) : n(n)
    {
        adj.resize(n + 1);
        capacity.assign(n + 1, vector<int>(n
            + 1, 0));
        level.resize(n + 1);
```

```
    ptr.resize(n + 1);
}
void add_edge(int u, int v, int cap)
{
    capacity[u][v] += cap;
    adj[u].push_back(v);
    adj[v].push_back(u);
}
bool bfs(int s, int t)
{
    fill(level.begin(), level.end(), -1);
    level[s] = 0;
    queue<int> q;
    q.push(s);

    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int v : adj[u])
        {
            if (level[v] == -1 &&
                capacity[u][v] > 0)
            {
                level[v] = level[u] + 1;
                q.push(v);
            }
        }
    }
    return level[t] != -1;
}
int dfs(int u, int t, int flow)
{
    if (u == t || flow == 0)
        return flow;
    for (int &i = ptr[u]; i <
        adj[u].size(); i++)
    {
        int v = adj[u][i];
        if (level[v] == level[u] + 1 &&
            capacity[u][v] > 0)
        {
            int bottleneck = dfs(v, t,
                min(flow,
                capacity[u][v]));
            if (bottleneck > 0)
            {
                capacity[u][v] -=
                    bottleneck;
                capacity[v][u] +=
                    bottleneck;
                return bottleneck;
            }
        }
    }
    return 0;
}
int max_flow(int s, int t)
{
    int flow = 0;
    while (bfs(s, t))
    {
        fill(ptr.begin(), ptr.end(), 0);
        while (int new_flow = dfs(s, t,
            INF))
        {
            flow += new_flow;
        }
    }
    return flow;
}
};
```

4.2 Edmonds Karp [60 lines]

```
//  O(V*E^2)
struct EdmondsKarp
{
    vector<vector<int>> capacity, adj;
    int n;

    EdmondsKarp(int n) : n(n)
    {
        capacity.assign(n + 1, vector<int>(n
            + 1, 0));
        adj.resize(n + 1);
    }
    void add_edge(int u, int v, int cap)
    {
        capacity[u][v] += cap;
        adj[u].push_back(v);
        adj[v].push_back(u); // Reverse edge
    }
    int bfs(int s, int t, vector<int> &parent)
    {
        fill(parent.begin(), parent.end(),
            -1);
        parent[s] = -2;
        queue<pair<int, int>> q;
        q.push({s, INF});

        while (!q.empty())
        {
            auto [u, flow] = q.front();
            q.pop();
            for (int v : adj[u])
            {
                if (parent[v] == -1 &&
                    capacity[u][v] > 0)
                {
                    parent[v] = u;
                    int new_flow = min(flow,
                        capacity[u][v]);
                    if (v == t) return
                        new_flow;
                    q.push({v, new_flow});
                }
            }
        }
        return 0;
    }
    int max_flow(int s, int t)
    {
        int flow = 0, new_flow;
        vector<int> parent(n + 1);
        while ((new_flow = bfs(s, t, parent))
            > 0)
        {
            flow += new_flow;
            int cur = t;
            while (cur != s)
            {
                int prev = parent[cur];
                capacity[prev][cur] -=
                    new_flow;
                capacity[cur][prev] +=
                    new_flow;
                cur = prev;
            }
        }
        return flow;
    }
};
```

4.3 Ford Fulkerson [50 lines]

```
// O(E*F),  where F is the maximum flow value.
struct FordFulkerson
{
    vector<vector<int>> capacity, adj;
    vector<bool> visited;
    int n;

    FordFulkerson(int n) : n(n)
    {
        capacity.assign(n + 1, vector<int>(n
            + 1, 0));
        adj.resize(n + 1);
    }
    void add_edge(int u, int v, int cap)
    {
        capacity[u][v] += cap;
        adj[u].push_back(v);
        adj[v].push_back(u); // Reverse edge
            for residual graph
    }
    int dfs(int u, int t, int flow)
    {
        if (u == t)
            return flow;
        visited[u] = true;
        for (int v : adj[u])
        {
            if (!visited[v] && capacity[u][v]
                > 0)
            {
                int bottleneck = dfs(v, t,
                    min(flow,
                    capacity[u][v]));
                if (bottleneck > 0)
                {
                    capacity[u][v] -=
                        bottleneck;
                    capacity[v][u] +=
                        bottleneck;
                    return bottleneck;
                }
            }
        }
        return 0;
    }
    int max_flow(int s, int t)
    {
        int flow = 0, new_flow;
        do
        {
            visited.assign(n + 1, false);
            new_flow = dfs(s, t, INF);
            flow += new_flow;
        } while (new_flow > 0);
        return flow;
    }
};
```

4.4 Hopcroft Karp [78 lines]

```
//O(E*root(V))
class HopcroftKarp
{
public:
    int n, m;
    vector<vector<int>> adj;
    vector<int> pairU, pairV, dist;
    HopcroftKarp(int n, int m) : n(n), m(m)
    {
        adj.resize(n + 1);
        pairU.assign(n + 1, 0);
        pairV.assign(m + 1, 0);
        dist.assign(n + 1, 0);
    }
    void addEdge(int u, int v)
    {
        adj[u].push_back(v);
    }
    bool bfs()
    {
        queue<int> q;
        for (int u = 1; u <= n; ++u)
        {
            if (pairU[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[pairV[v]] ==
                        INT_MAX)
                    {
                        dist[pairV[v]] =
                            dist[u] + 1;
                        q.push(pairV[v]);
                    }
                }
            }
        }
        return dist[0] != INT_MAX;
    }
    bool dfs(int u)
    {
        if (u == 0) return true;
        for (int v : adj[u])
        {
            if (dist[pairV[v]] == dist[u] + 1
                && dfs(pairV[v]))
            {
```

```
                pairV[v] = u;
                pairU[u] = v;
                return true;
            }
        }
        dist[u] = INT_MAX;
        return false;
    }

    int maxMatching()
    {
        int matching = 0;
        while (bfs())
        {
            for (int u = 1; u <= n; ++u)
                if (pairU[u] == 0 && dfs(u))
                    matching++;
        }
        return matching;
    }
};
```

4.5 Max Flow Path [99 lines]

```
struct MaxFlowpath
{
    struct Edge
    {
        int v, rev, cap, flow;
    };
    int n;
    vector<vector<Edge>> adj;

    MaxFlowpath(int n) : n(n)
    {
        adj.resize(n + 1);
    }
    void add_edge(int u, int v)
    {
        adj[u].push_back({v,
            (int)adj[v].size(), 1, 0});
        adj[v].push_back({u,
            (int)adj[u].size() - 1, 0, 0});
    }
    bool bfs(int s, int t, vector<int>
        &parent, vector<int> &edge_index)
    {
        vector<bool> visited(n + 1, false);
        queue<int> q;
        q.push(s);
        visited[s] = true;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i <
                adj[u].size(); i++)
            {
                Edge &e = adj[u][i];
                if (!visited[e.v] && e.cap >
                    0)
                {
                    visited[e.v] = true;
                    parent[e.v] = u;
                    edge_index[e.v] = i;
                    if (e.v == t) return true;
                    q.push(e.v);
                }
            }
        }
        return false;
    }
    int max_flow(int s, int t)
    {
        int total_flow = 0;
        while (true)
        {
            vector<int> parent(n + 1, -1),
                edge_index(n + 1, -1);
            if (!bfs(s, t, parent,
                edge_index))
                break;
            int push_flow = INT_MAX, cur = t;
            while (cur != s)
            {
                int prev = parent[cur];
                Edge &e =
                    adj[prev][edge_index[cur]];

                push_flow = min(push_flow,
                    e.cap);
                cur = prev;
            }
            cur = t;
            while (cur != s)
            {
                int prev = parent[cur];
                Edge &e =
                    adj[prev][edge_index[cur]];

                e.cap -= push_flow;
                e.flow += push_flow;
                adj[cur][e.rev].cap +=
                    push_flow;
                adj[cur][e.rev].flow -=
                    push_flow;
                cur = prev;
            }
            total_flow += push_flow;
        }
        return total_flow;
    }
    void dfs(int u, vector<int> &p)
    {
        p.push_back(u);
        for (auto &e : adj[u])
        {
            if (e.flow > 0)
            {
                e.flow--;
                dfs(e.v, p);
                break;
            }
        }
    }
    vector<vector<int>>
        find_disjoint_paths(int s, int f)
    {
        vector<vector<int>> paths;
        while (f--)
        {
            vector<int> path;
            dfs(s, path);
            paths.push_back(path);
        }
        return paths;
    }
};
```

4.6 Max Flow [93 lines]

```
struct MaxFlow
{
    struct Edge
    {
        int v, rev, cap;
    };
    int n;
    vector<vector<Edge>> adj;
    MaxFlow(int n) : n(n)
    {
        adj.resize(n + 1);
    }
    void add_edge(int u, int v)
    {
        adj[u].push_back({v,
            (int)adj[v].size(), 1});
        adj[v].push_back({u,
            (int)adj[u].size() - 1, 0});
    }
    bool bfs(int s, int t, vector<int>
        &parent, vector<int> &edge_index)
    {
        vector<bool> visited(n + 1, false);
        queue<int> q;
        q.push(s);
        visited[s] = true;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i <
                adj[u].size(); i++)
            {
                Edge &e = adj[u][i];
                if (!visited[e.v] && e.cap >
                    0)
                {
                    visited[e.v] = true;
                    parent[e.v] = u;
                    edge_index[e.v] = i;
                    if (e.v == t) return true;
                    q.push(e.v);
                }
            }
        }
        return false;
    }
    int max_flow(int s, int t)
    {
        int total_flow = 0;
        while (true)
        {
            vector<int> parent(n + 1, -1),
                edge_index(n + 1, -1);
            if (!bfs(s, t, parent,
                edge_index))
                break;
            int push_flow = INT_MAX, cur = t;
            while (cur != s)
            {
                int prev = parent[cur];
                Edge &e =
                    adj[prev][edge_index[cur]];

                push_flow = min(push_flow,
                    e.cap);
                cur = prev;
            }
            cur = t;
            while (cur != s)
            {
                int prev = parent[cur];
                Edge &e =
                    adj[prev][edge_index[cur]];

                e.cap -= push_flow;
                adj[cur][e.rev].cap +=
                    push_flow;
                cur = prev;
            }
            total_flow += push_flow;
        }
        return total_flow;
    }
    void dfs(int u, vector<bool> &visited)
    {
        visited[u] = true;
        for (const auto &e : adj[u])
        {
            if (e.cap > 0 && !visited[e.v])
                dfs(e.v, visited);
        }
    }
    vector<pair<int, int>> find_min_cut(int s)
    {
        vector<bool> visited(n + 1, false);
        dfs(s, visited);
        vector<pair<int, int>> cut_edges;
        for (int u = 1; u <= n; u++)
        {
            if (!visited[u]) continue;
            for (const auto &e : adj[u])
                if (!visited[e.v] && e.cap ==
                    0)
                    cut_edges.emplace_back(u,
                    e.v);
        }
        return cut_edges;
    }
};
```

4.7 Min Cost Max Flow [87 lines]

```
struct MinCostMaxFlow
{
    struct Edge
    {
        int v, rev;    // Destination and
            index of the reverse edge
        int cap, cost; // Capacity and cost of
            the edge
    };
    vector<vector<Edge>> adj;
```

```
int n;
MinCostMaxFlow(int n) : n(n)
{
    adj.resize(n + 1); // 1-based indexing
}
void add_edge(int u, int v, int cap, int
    cost)
{
    adj[u].push_back({v,
        (int)adj[v].size(), cap, cost});
    adj[v].push_back({u,
        (int)adj[u].size() - 1, 0,
        -cost});
}
bool spfa(int s, int t, vector<int>
    &parent, vector<int> &edge_index,
    vector<int> &dist)
{
    dist.assign(n + 1, INF);
    parent.assign(n + 1, -1);
    edge_index.assign(n + 1, -1);
    vector<bool> in_queue(n + 1, false);
    queue<int> q;

    dist[s] = 0;
    q.push(s);
    in_queue[s] = true;

    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        in_queue[u] = false;
        for (int i = 0; i <
            adj[u].size(); i++)
        {
            Edge &e = adj[u][i];
            if (e.cap > 0 && dist[u] +
                e.cost < dist[e.v])
            {
                dist[e.v] = dist[u] +
                    e.cost;
                parent[e.v] = u;
                edge_index[e.v] = i;
                if (!in_queue[e.v])
                {
                    q.push(e.v);
                    in_queue[e.v] = true;
                }
            }
        }
    }
    return dist[t] != INF;
}

pair<int, int> max_flow(int s, int t)
{
    int total_flow = 0, total_cost = 0;
    vector<int> parent, edge_index, dist;

    while (spfa(s, t, parent, edge_index,
        dist))
    {
        // Find the bottleneck capacity
            along the path
        int push_flow = INF, cur = t;
        while (cur != s)
        {
            int prev = parent[cur];
            Edge &e =
                adj[prev][edge_index[cur]];
            push_flow = min(push_flow,
                e.cap);
            cur = prev;
        }
        // Update the residual graph along
            the path
        cur = t;
        while (cur != s)
        {
            int prev = parent[cur];
            Edge &e =
                adj[prev][edge_index[cur]];
            e.cap -= push_flow;
            adj[e.v][e.rev].cap +=
                push_flow;
            cur = prev;
        }
        total_flow += push_flow;
        total_cost += push_flow * dist[t];
    }

    return {total_flow, total_cost};
}
};
```

5 Game Theory

5.1 Points to be noted [14 lines]

```
>[First Write a Brute Force solution]
>Nim = all xor
>Misere Nim = Nim + corner case: if all piles
    are 1, reverse(nim)
>Bogus Nim = Nim
>Staircase Nim = Odd indexed pile Nim (Even
    indexed pile doesnt matter, as one player
    can give bogus moves to drop all even
    piles to ground)
>Sprague Grundy: [Every impartial game under
    the normal play convention is equivalent
    to a one-heap game of nim]
Every tree = one nim pile = tree root value;
    tree leaf value = 0; tree node value = mex
    of all child nodes.
[Careful: one tree node can become multiple
    new tree roots(multiple elements in one
    node), then the value of that node = xor
    of all those root values]
>Hackenbush(Given a rooted tree; cut an edge
    in one move; subtree under that edge gets
    removed; last player to cut wins):
 Colon:
    //G(u) = (G(v1)+1)⊕(G(v2)+1)⊕···[v1, v2, ···
    are childs of u]
For multiple trees ans is their xor
>Hackenbush on graph (instead of tree given an
    rooted graph):
fusion: All edges in a cycle can be fused to
    get a tree structure; build a super node,
    connect some single nodes with that super
    node, number of single nodes is the number
    of edges in the cycle.
Sol: [Bridge component tree] mark all
    bridges, a group of edges that are not
    bridges, becomes one component and
    contributes number of edges to the
    hackenbush. (even number of edges
    contributes 0, odd number of edges
    contributes 1)
```

6 Geometry

6.1 Convex Hull [60 lines]

```
struct pt {
double x, y;
pr(){}
pt(double x,double y)
{
this->x=x;
this->y=y;
}
bool operator == (pt const& t) const {
return x == t.x && y == t.y;
}
};
int orientation(pt a, pt b, pt c) {
double v =
a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
if (v < 0) return -1; // clockwise
if (v > 0) return +1; // counter-clockwise
return 0;
}
bool cw(pt a, pt b, pt c, bool
    include_collinear) {
int o = orientation(a, b, c);
return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return
orientation(a, b, c) == 0; }
void convex_hull(vector<pt>& a, bool
    include_collinear
= false) {
pt p0 = *min_element(a.begin(), a.end(), [](pt
    a, pt
b) {
return make_pair(a.y, a.x) < make_pair(b.y,
b.x);
});
sort(a.begin(), a.end(), [&p0](const pt& a,
    const
pt& b) {
int o = orientation(p0, a, b);
if (o == 0)
return (p0.x-a.x)*(p0.x-a.x) +
(p0.y-a.y)*(p0.y-a.y)
< (p0.x-b.x)*(p0.x-b.x) +
(p0.y-b.y)*(p0.y-b.y);
return o < 0;
});
if (include_collinear) {
int i = (int)a.size()-1;
while (i >= 0 && collinear(p0, a[i],
    a.back()))
i--;
reverse(a.begin()+i+1, a.end());
}
vector<pt> st;
for (int i = 0; i < (int)a.size(); i++) {
while (st.size() > 1 && !cw(st[st.size()-2],
st.back(), a[i], include_collinear))
st.pop_back();
st.pop_back(a[i]);
}
if (include_collinear == false && st.size()
    == 2 &&
st[0] == st[1])
st.pop_back();
a = st;
}
```

6.2 Geometry [891 lines]

```
int sign(T x) { return (x > eps) - (x <
    -eps); }
struct PT {
  T x, y;
  PT() { x = 0, y = 0; }
  PT(T x, T y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)  {}
  PT operator + (const PT &a) const { return
      PT(x + a.x, y + a.y); }
  PT operator - (const PT &a) const { return
      PT(x - a.x, y - a.y); }
  PT operator * (const T a) const { return
      PT(x * a, y * a); }
  friend PT operator * (const T &a, const PT
      &b) { return PT(a * b.x, a * b.y); }
  PT operator / (const T a) const { return
      PT(x / a, y / a); }
  bool operator == (PT a) const { return
      sign(a.x - x) == 0 && sign(a.y - y) ==
      0; }
  bool operator != (PT a) const { return
      !(*this == a); }
  bool operator < (PT a) const { return
      sign(a.x - x) == 0 ? y < a.y : x < a.x;
      }
  bool operator > (PT a) const { return
      sign(a.x - x) == 0 ? y > a.y : x > a.x;
      }
  T norm() { return sqrt(x * x + y * y); }
  T norm2() { return x * x + y * y; }
  PT perp() { return PT(-y, x); }
  T arg() { return atan2(y, x); }
  PT truncate(T r) { // returns a vector with
      norm r and having same direction
    T k = norm();
    if (!sign(k)) return *this;
    r /= k;
    return PT(x * r, y * r);
  }
};
istream &operator >> (istream &in, PT &p) {
    return in >> p.x >> p.y; }
ostream &operator << (ostream &out, PT &p) {
    return out << "(" << p.x << "," << p.y <<
    ")"; }
inline T dot(PT a, PT b) { return a.x * b.x +
    a.y * b.y; }
```

```
inline T dist2(PT a, PT b) { return dot(a -
    b, a - b); }
inline T dist(PT a, PT b) { return sqrt(dot(a
    - b, a - b)); }
inline T cross(PT a, PT b) { return a.x * b.y
    - a.y * b.x; }
inline T cross2(PT a, PT b, PT c) { return
    cross(b - a, c - a); }
inline int orientation(PT a, PT b, PT c) {
    return sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, T t) { return PT(a.x *
    cos(t) - a.y * sin(t), a.x * sin(t) + a.y
    * cos(t)); }
PT rotatecw(PT a, T t) { return PT(a.x *
    cos(t) + a.y * sin(t), -a.x * sin(t) + a.y
    * cos(t)); }
T rad_to_deg(T r) { return (r * 180.0 / PI); }
T deg_to_rad(T d) { return (d * PI / 180.0); }
T get_angle(PT a, PT b) {
  T costheta = dot(a, b) / a.norm() /
      b.norm();
  return acos(max((T)-1.0, min((T)1.0,
      costheta)));
}
bool is_point_in_angle(PT b, PT a, PT c, PT p)
    { // does point p lie in angle <bac
  assert(orientation(a, b, c) != 0);
  if (orientation(a, c, b) < 0) swap(b, c);
  return orientation(a, c, p) >= 0 &&
      orientation(a, b, p) <= 0;
}
bool half(PT p) {
  return p.y > 0.0 || (p.y == 0.0 && p.x <
      0.0);
}
void polar_sort(vector<PT> &v) { // sort
    points in counterclockwise
  sort(v.begin(), v.end(), [](PT a,PT b) {
    return make_tuple(half(a), 0.0, a.norm2())
        < make_tuple(half(b), cross(a, b),
        b.norm2());
  });
}
void polar_sort(vector<PT> &v, PT o) { // sort
    points in counterclockwise with respect to
    point o
  sort(v.begin(), v.end(), [&](PT a,PT b) {
    return make_tuple(half(a - o), 0.0, (a -
        o).norm2()) < make_tuple(half(b - o),
        cross(a - o, b - o), (b -
        o).norm2());
  });
}
struct line {
  PT a, b; // goes through points a and b
  PT v; T c;  //line form: direction vec
      [cross] (x, y) = c
  line() {}
  //direction vector v and offset c
  line(PT v, T c) : v(v), c(c) {
    auto p = get_points();
    a = p.first; b = p.second;
  }
  // equation ax + by + c = 0
  line(T _a, T _b, T _c) : v({_b, -_a}),
      c(-_c) {
  auto p = get_points();
    a = p.first; b = p.second;
  }
  // goes through points p and q
  line(PT p, PT q) : v(q - p), c(cross(v,
      p)), a(p), b(q) {}
    pair<PT, PT> get_points() { //extract any
        two points from this line
  PT p, q; T a = -v.y, b = v.x; // ax + by = c
  if (sign(a) == 0) {
    p = PT(0, c / b);
    q = PT(1, c / b);
  }
  else if (sign(b) == 0) {
    p = PT(c / a, 0);
    q = PT(c / a, 1);
  }
  else {
    p = PT(0, c / b);
    q = PT(1, (c - a) / b);
  }
  return {p, q};
    }
  // ax + by + c = 0
  array<T, 3> get_abc() {
    T a = -v.y, b = v.x;
    return {a, b, -c};
  }
  // 1 if on the left, -1 if on the right, 0 if
      on the line
  int side(PT p) { return sign(cross(v, p) -
      c); }
  // line that is perpendicular to this and
      goes through point p
  line perpendicular_through(PT p) { return
      {p, p + perp(v)}; }
  // translate the line by vector t i.e.
      shifting it by vector t
  line translate(PT t) { return {v, c +
      cross(v, t)}; }
  // compare two points by their orthogonal
      projection on this line
  // a projection point comes before another if
      it comes first according to vector v
  bool cmp_by_projection(PT p, PT q) { return
      dot(v, p) < dot(v, q); }
  line shift_left(T d) {
  PT z = v.perp().truncate(d);
  return line(a + z, b + z);
  }
};
// find a point from a through b with distance
    d
PT point_along_line(PT a, PT b, T d) {
  assert(a != b);
  return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b
    assuming a != b
PT project_from_point_to_line(PT a, PT b, PT
    c) {
  return a + (b - a) * dot(c - a, b - a) / (b
      - a).norm2();
}
// reflection point c onto line through a and b
    assuming a != b
PT reflection_from_point_to_line(PT a, PT b,
    PT c) {
  PT p = project_from_point_to_line(a,b,c);
  return p + p - c;
}
// minimum distance from point c to line
    through a and b
T dist_from_point_to_line(PT a, PT b, PT c) {
  return fabs(cross(b - a, c - a) / (b -
      a).norm());
}
// returns true if  point p is on line segment
    ab
bool is_point_on_seg(PT a, PT b, PT p) {
  if (fabs(cross(p - b, a - b)) < eps) {
    if (p.x < min(a.x, b.x) - eps || p.x >
        max(a.x, b.x) + eps) return false;
    if (p.y < min(a.y, b.y) - eps || p.y >
        max(a.y, b.y) + eps) return false;
    return true;
  }
  return false;
}
// minimum distance point from point c to
    segment ab that lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c)
    {
  T r = dist2(a, b);
  if (sign(r) == 0) return a;
  r = dot(c - a, b - a) / r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b - a) * r;
}
// minimum distance from point c to segment ab
T dist_from_point_to_seg(PT a, PT b, PT c) {
  return dist(c, project_from_point_to_seg(a,
      b, c));
}
// 0 if not parallel, 1 if parallel, 2 if
    collinear
int is_parallel(PT a, PT b, PT c, PT d) {
  T k = fabs(cross(b - a, d - c));
  if (k < eps){
    if (fabs(cross(a - b, a - c)) < eps &&
        fabs(cross(c - d, c - a)) < eps)
        return 2;
    else return 1;
  }
  else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
  if (fabs(cross(a - c, c - d)) < eps &&
      fabs(cross(b - c, c - d)) < eps) return
      true;
  return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
  PT p = a - b, q = c - b;
  return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is
    cw to the line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
  int c = sign(cross(p - a, b - a));
  if (c < 0) return 1;
  if (c > 0) return 2;
  return 3;
}
// intersection point between ab and cd
    assuming unique intersection exists
bool line_line_intersection(PT a, PT b, PT c,
    PT d, PT &ans) {
  T a1 = a.y - b.y, b1 = b.x - a.x, c1 =
      cross(a, b);
  T a2 = c.y - d.y, b2 = d.x - c.x, c2 =
      cross(c, d);
  T det = a1 * b2 - a2 * b1;
  if (det == 0) return 0;
  ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2
      - a1 * c2) / det);
  return 1;
}
// intersection point between segment ab and
    segment cd assuming unique intersection
    exists
bool seg_seg_intersection(PT a, PT b, PT c, PT
    d, PT &ans) {
  T oa = cross2(c, d, a), ob = cross2(c, d,
      b);
  T oc = cross2(a, b, c), od = cross2(a, b,
      d);
  if (oa * ob < 0 && oc * od < 0){
    ans = (a * ob - b * oa) / (ob - oa);
    return 1;
  }
  else return 0;
}
// intersection point between segment ab and
    segment cd assuming unique intersection may
    not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT
    b,  PT c,  PT d) {
  PT ans;
  if (seg_seg_intersection(a, b, c, d, ans))
      return {ans};
  set<PT> se;
  if (is_point_on_seg(c, d, a)) se.insert(a);
  if (is_point_on_seg(c, d, b)) se.insert(b);
  if (is_point_on_seg(a, b, c)) se.insert(c);
  if (is_point_on_seg(a, b, d)) se.insert(d);
  return se;
}
// intersection  between segment ab and line
    cd
// 0 if do not intersect, 1 if proper
    intersect, 2 if segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d)
    {
  T p = cross2(c, d, a);
```

```
  T q = cross2(c, d, b);
  if (sign(p) == 0 && sign(q) == 0) return 2;
  else if (p * q < 0) return 1;
  else return 0;
}
// intersection between segament ab and line cd
    assuming unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c,
    PT d, PT &ans) {
  bool k = seg_line_relation(a, b, c, d);
  assert(k != 2);
  if (k) line_line_intersection(a, b, c, d,
      ans);
  return k;
}
// minimum distance from segment ab to segment
    cd
T dist_from_seg_to_seg(PT a, PT b, PT c, PT d)
    {
  PT dummy;
  if (seg_seg_intersection(a, b, c, d, dummy))
      return 0.0;
  else return min({dist_from_point_to_seg(a,
      b, c), dist_from_point_to_seg(a, b, d),
    dist_from_point_to_seg(c, d, a),
      dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray
    (starting point a and direction vector b)
T dist_from_point_to_ray(PT a, PT b, PT c) {
  b = a + b;
  T r = dot(c - a, b - a);
  if (r < 0.0) return dist(c, a);
  return dist_from_point_to_line(a, b, c);
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT
    bs, PT bd) {
  T dx = bs.x - as.x, dy = bs.y - as.y;
  T det = bd.x * ad.y - bd.y * ad.x;
  if (fabs(det) < eps) return 0;
  T u = (dy * bd.x - dx * bd.y) / det;
  T v = (dy * ad.x - dx * ad.y) / det;
  if (sign(u) >= 0 && sign(v) >= 0) return 1;
  else return 0;
}
T ray_ray_distance(PT as, PT ad, PT bs, PT bd)
    {
  if (ray_ray_intersection(as, ad, bs, bd))
      return 0.0;
  T ans = dist_from_point_to_ray(as, ad, bs);
  ans = min(ans, dist_from_point_to_ray(bs,
      bd, as));
  return ans;
}
struct circle {
  PT p; T r;
  circle() {}
  circle(PT _p, T _r): p(_p), r(_r) {};
  // center (x, y) and radius r
  circle(T x, T y, T _r): p(PT(x, y)), r(_r)
      {};
  // circumcircle of a triangle
  // the three points must be unique
  circle(PT a, PT b, PT c) {
    b = (a + b) * 0.5;
    c = (a + c) * 0.5;
    line_line_intersection(b, b + rotatecw90(a
        - b), c, c + rotatecw90(a - c), p);
    r = dist(a, p);
  }
  // inscribed circle of a triangle
  // pass a bool just to differentiate from
      circumcircle
  circle(PT a, PT b, PT c, bool t) {
    line u, v;
    T m = atan2(b.y - a.y, b.x - a.x), n =
        atan2(c.y - a.y, c.x - a.x);
    u.a = a;
    u.b = u.a + (PT(cos((n + m)/2.0), sin((n
        + m)/2.0)));
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x), n =
        atan2(c.y - b.y, c.x - b.x);
    v.b = v.a + (PT(cos((n + m)/2.0), sin((n
        + m)/2.0)));
    line_line_intersection(u.a, u.b, v.a,
        v.b, p);
    r = dist_from_point_to_seg(a, b, p);
  }
  bool operator == (circle v) { return p ==
      v.p && sign(r - v.r) == 0; }
  T area() { return PI * r * r; }
  T circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if
    inside circle
int circle_point_relation(PT p, T r, PT b) {
  T d = dist(p, b);
  if (sign(d - r) < 0) return 2;
  if (sign(d - r) == 0) return 1;
  return 0;
}
// 0 if outside, 1 if on circumference, 2 if
    inside circle
int circle_line_relation(PT p, T r, PT a, PT
    b) {
  T d = dist_from_point_to_line(a, b, p);
  if (sign(d - r) < 0) return 2;
  if (sign(d - r) == 0) return 1;
  return 0;
}
//compute intersection of line through points a
    and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, T
    r, PT a, PT b) {
  vector<PT> ret;
  b = b - a; a = a - c;
  T A = dot(b, b), B = dot(a, b);
  T C = dot(a, a) - r * r, D = B * B - A * C;
  if (D < -eps) return ret;
  ret.push_back(c + a + b * (-B + sqrt(D +
      eps)) / A);
  if (D > eps) ret.push_back(c + a + b * (-B
      - sqrt(D)) / A);
  return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, T r, PT b, T
    R) {
  T d = dist(a, b);
  if (sign(d - r - R) > 0)  return 5;
  if (sign(d - r - R) == 0) return 4;
  T l = fabs(r - R);
  if (sign(d - r - R) < 0 && sign(d - l) > 0)
      return 3;
  if (sign(d - l) == 0) return 2;
  if (sign(d - l) < 0) return 1;
  assert(0); return -1;
}
vector<PT> circle_circle_intersection(PT a, T
    r, PT b, T R) {
  if (a == b && sign(r - R) == 0) return
      {PT(1e18, 1e18)};
  vector<PT> ret;
  T d = sqrt(dist2(a,  b));
  if (d > r + R || d + min(r,  R) < max(r,
      R)) return ret;
  T x = (d * d - R * R + r * r) / (2 * d);
  T y = sqrt(r * r - x * x);
  PT v = (b - a) / d;
  ret.push_back(a + v * x  +  rotateccw90(v)
      * y);
  if (y > 0) ret.push_back(a + v * x -
      rotateccw90(v) * y);
  return ret;
}
// returns two circle c1, c2 through points a,
    b and of radius r
// 0 if there is no such circle, 1 if one
    circle, 2 if two circle
int get_circle(PT a, PT b, T r, circle &c1,
    circle &c2) {
  vector<PT> v =
      circle_circle_intersection(a, r, b, r);
  int t = v.size();
  if (!t) return 0;
  c1.p = v[0], c1.r = r;
  if (t == 2) c2.p = v[1], c2.r = r;
  return t;
}
// returns two circle c1, c2 which is tangent
    to line u,  goes through
// point q and has radius r1; 0 for no circle,
    1 if c1 = c2 , 2 if c1 != c2
int get_circle(line u, PT q, T r1, circle
    &c1, circle &c2) {
  T d = dist_from_point_to_line(u.a, u.b, q);
  if (sign(d - r1 * 2.0) > 0) return 0;
  if (sign(d) == 0) {
    cout << u.v.x << ' ' << u.v.y << '\n';
    c1.p = q + rotateccw90(u.v).truncate(r1);
    c2.p = q + rotatecw90(u.v).truncate(r1);
    c1.r = c2.r = r1;
    return 2;
  }
  line u1 = line(u.a +
      rotateccw90(u.v).truncate(r1), u.b +
      rotateccw90(u.v).truncate(r1));
  line u2 = line(u.a +
      rotatecw90(u.v).truncate(r1), u.b +
      rotatecw90(u.v).truncate(r1));
  circle cc = circle(q, r1);
  PT p1, p2; vector<PT> v;
  v = circle_line_intersection(q, r1, u1.a,
      u1.b);
  if (!v.size()) v =
      circle_line_intersection(q, r1, u2.a,
      u2.b);
  v.push_back(v[0]);
  p1 = v[0], p2 = v[1];
  c1 = circle(p1, r1);
  if (p1 == p2) {
    c2 = c1;
    return 1;
  }
  c2 = circle(p2, r1);
  return 2;
}
// returns area of intersection between two
    circles
T circle_circle_area(PT a, T r1, PT b, T r2) {
  T d = (a - b).norm();
  if(r1 + r2 < d + eps) return 0;
  if(r1 + d < r2 + eps) return PI * r1 * r1;
  if(r2 + d < r1 + eps) return PI * r2 * r2;
  T theta_1 = acos((r1 * r1 + d * d - r2 * r2)
      / (2 * r1 * d)),
    theta_2 = acos((r2 * r2 + d * d - r1 *
        r1)/(2 * r2 * d));
  return r1 * r1 * (theta_1 - sin(2 *
      theta_1)/2.) + r2 * r2 * (theta_2 -
      sin(2 * theta_2)/2.);
}
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, T r, PT q,
    line &u, line &v) {
  int x = sign(dist2(p, q) - r * r);
  if (x < 0) return 0; // point in cricle
  if (x == 0) { // point on circle
    u = line(q, q + rotateccw90(q - p));
    v = u;
    return 1;
  }
  T d = dist(p, q);
  T l = r * r / d;
  T h = sqrt(r * r - l * l);
  u = line(q, p + ((q - p).truncate(l) +
      (rotateccw90(q - p).truncate(h))));
  v = line(q, p + ((q - p).truncate(l) +
      (rotatecw90(q - p).truncate(h))));
  return 2;
}
// returns outer tangents line of two circles
// if inner == 1 it returns inner tangent
    lines
int tangents_lines_from_circle(PT c1, T r1, PT
    c2, T r2, bool inner, line &u, line &v) {
  if (inner) r2 = -r2;
  PT d = c2 - c1;
  T dr = r1 - r2, d2 = d.norm2(), h2 = d2 - dr
      * dr;
  if (d2 == 0 || h2 < 0) {
    assert(h2 != 0);
    return 0;
  }
```

```
  vector<pair<PT, PT>>out;
  for (int tmp: {- 1, 1}) {
    PT v = (d * dr + rotateccw90(d) * sqrt(h2)
        * tmp) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  u = line(out[0].first, out[0].second);
  if (out.size() == 2) v = line(out[1].first,
      out[1].second);
  return 1 + (h2 > 0);
}
// -1 if strictly inside, 0 if on the polygon,
    1 if strictly outside
int is_point_in_triangle(PT a, PT b, PT c, PT
    p) {
  if (sign(cross(b - a,c - a)) < 0) swap(b,
      c);
  int c1 = sign(cross(b - a,p - a));
  int c2 = sign(cross(c - b,p - b));
  int c3 = sign(cross(a - c,p - c));
  if (c1<0 || c2<0 || c3 < 0) return 1;
  if (c1 + c2 + c3 != 3) return 0;
  return -1;
}
T perimeter(vector<PT> &p) {
  T ans=0; int n = p.size();
  for (int i = 0; i < n; i++) ans +=
      dist(p[i], p[(i + 1) % n]);
  return ans;
}
T area(vector<PT> &p) {
  T ans = 0; int n = p.size();
  for (int i = 0; i < n; i++) ans +=
      cross(p[i], p[(i + 1) % n]);
  return fabs(ans) * 0.5;
}
// centroid of a (possibly non-convex) polygon,

// assuming that the coordinates are listed in
    a clockwise or
// counterclockwise fashion.  Note that the
    centroid is often known as
// the "center of gravity" or "center of
    mass".
PT centroid(vector<PT> &p) {
  int n = p.size(); PT c(0, 0);
  T sum = 0;
  for (int i = 0; i < n; i++) sum +=
      cross(p[i], p[(i + 1) % n]);
  T scale = 3.0 * sum;
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    c = c + (p[i] + p[j]) * cross(p[i], p[j]);
  }
  return c / scale;
}
// 0 if cw, 1 if ccw
bool get_direction(vector<PT> &p) {
  T ans = 0; int n = p.size();
  for (int i = 0; i < n; i++) ans +=
      cross(p[i], p[(i + 1) % n]);
  if (sign(ans) > 0) return 1;
  return 0;
}
// it returns a point such that the sum of
    distances
// from that point to all points in p  is
    minimum
// O(n log^2 MX)
PT geometric_median(vector<PT> p) {
  auto tot_dist = [&](PT z) {
    T res = 0;
    for (int i = 0; i < p.size(); i++) res +=
        dist(p[i], z);
    return res;
  };
  auto findY = [&](T x) {
    T yl = -1e5, yr = 1e5;
    for (int i = 0; i < 60; i++) {
      T ym1 = yl + (yr - yl) / 3;
      T ym2 = yr - (yr - yl) / 3;
      T d1 = tot_dist(PT(x, ym1));
      T d2 = tot_dist(PT(x, ym2));
      if (d1 < d2) yr = ym2;
      else yl = ym1;
    }
    return pair<T, T> (yl, tot_dist(PT(x,
        yl)));
  };
  T xl = -1e5, xr = 1e5;
  for (int i = 0; i < 60; i++) {
    T xm1 = xl + (xr - xl) / 3;
    T xm2 = xr - (xr - xl) / 3;
    T y1, d1, y2, d2;
    auto z = findY(xm1); y1 = z.first; d1 =
        z.second;
    z = findY(xm2); y2 = z.first; d2 =
        z.second;
    if (d1 < d2) xr = xm2;
    else xl = xm1;
  }
  return {xl, findY(xl).first };
}
vector<PT> convex_hull(vector<PT> &p) {
  if (p.size() <= 1) return p;
  vector<PT> v = p;
  sort(v.begin(), v.end());
  vector<PT> up, dn;
  for (auto& p : v) {
    while (up.size() > 1 &&
        orientation(up[up.size() - 2],
        up.back(), p) >= 0) {
      up.pop_back();
    }
    while (dn.size() > 1 &&
        orientation(dn[dn.size() - 2],
        dn.back(), p) <= 0) {
      dn.pop_back();
    }
    up.push_back(p);
    dn.push_back(p);
  }
  v = dn;
  if (v.size() > 1) v.pop_back();
  reverse(up.begin(), up.end());
  up.pop_back();
  for (auto& p : up) {
    v.push_back(p);
  }
  if (v.size() == 2 && v[0] == v[1])
      v.pop_back();
  return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
  bool s[3]; s[0] = s[1] = s[2] = 0;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    int k = (j + 1) % n;
    s[sign(cross(p[j] - p[i], p[k] - p[i])) +
        1] = 1;
    if (s[0] && s[2]) return 0;
  }
  return 1;
}
// -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);
  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;
}
bool is_point_on_polygon(vector<PT> &p, const
    PT& z) {
  int n = p.size();
  for (int i = 0; i < n; i++) {
    if (is_point_on_seg(p[i], p[(i + 1) % n],
        z)) return 1;
  }
  return 0;
}
// returns 1e9 if the point is on the polygon
int winding_number(vector<PT> &p, const PT& z)
    { // O(n)
  if (is_point_on_polygon(p, z)) return 1e9;
  int n = p.size(), ans = 0;
  for (int i = 0; i < n; ++i) {
    int j = (i + 1) % n;
    bool below = p[i].y < z.y;
    if (below != (p[j].y < z.y)) {
      auto orient = orientation(z, p[j],
          p[i]);
      if (orient == 0) return 0;
      if (below == (orient > 0)) ans += below
          ? 1 : -1;
    }
  }
  return ans;
}
// -1 if strictly inside, 0 if on the polygon,
    1 if strictly outside
int is_point_in_polygon(vector<PT> &p, const
    PT& z) { // O(n)
  int k = winding_number(p, z);
  return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
// id of the vertex having maximum dot product
    with z
// polygon must need to be convex
// top - upper right vertex
// for minimum dot product negate z and return
    -dot(z, p[id])
int extreme_vertex(vector<PT> &p, const PT
    &z, const int top) { // O(log n)
  int n = p.size();
  if (n == 1) return 0;
  T ans = dot(p[0], z); int id = 0;
  if (dot(p[top], z) > ans) ans = dot(p[top],
      z), id = top;
  int l = 1, r = top - 1;
  while (l < r) {
    int mid = l + r >> 1;
    if (dot(p[mid + 1], z) >= dot(p[mid], z))
        l = mid + 1;
    else r = mid;
  }
  if (dot(p[l], z) > ans) ans = dot(p[l], z),
      id = l;
  l = top + 1, r = n - 1;
  while (l < r) {
    int mid = l + r >> 1;
    if (dot(p[(mid + 1) % n], z) >=
        dot(p[mid], z)) l = mid + 1;
    else r = mid;
  }
  l %= n;
  if (dot(p[l], z) > ans) ans = dot(p[l], z),
      id = l;
  return id;
}
// maximum distance from any point on the
    perimeter to another point on the
    perimeter
T diameter(vector<PT> &p) {
  int n = (int)p.size();
  if (n == 1) return 0;
  if (n == 2) return dist(p[0], p[1]);
  T ans = 0;
  int i = 0, j = 1;
  while (i < n) {
    while (cross(p[(i + 1) % n] - p[i], p[(j
        + 1) % n] - p[j]) >= 0) {
      ans = max(ans, dist2(p[i], p[j]));
      j = (j + 1) % n;
    }
    ans = max(ans, dist2(p[i], p[j]));
    i++;
  }
  return sqrt(ans);
}
// given n points, find the minimum enclosing
    circle of the points
// call convex_hull() before this for faster
    solution
```

```cpp
// expected O(n)
circle minimum_enclosing_circle(vector<PT> &p)
    {
  random_shuffle(p.begin(), p.end());
  int n = p.size();
  circle c(p[0], 0);
  for (int i = 1; i < n; i++) {
    if (sign(dist(c.p, p[i]) - c.r) > 0) {
      c = circle(p[i], 0);
      for (int j = 0; j < i; j++) {
        if (sign(dist(c.p, p[j]) - c.r) > 0) {
          c = circle((p[i] + p[j]) / 2,
              dist(p[i], p[j]) / 2);
          for (int k = 0; k < j; k++) {
            if (sign(dist(c.p, p[k]) - c.r) >
                0) {
              c = circle(p[i], p[j], p[k]);
            }
          }
        }
      }
    }
  }
  return c;
}
// not necessarily convex, boundary is included
    in the intersection
// returns total intersected length
// it returns the sum of the lengths of the
    portions of the line that are inside the
    polygon
T polygon_line_intersection(vector<PT> p, PT
    a, PT b) {
  int n = p.size();
  p.push_back(p[0]);
  line l = line(a, b);
  T ans = 0.0;
  vector< pair<T, int> > vec;
  for (int i = 0; i < n; i++) {
    int s1 = orientation(a, b, p[i]);
    int s2 = orientation(a, b, p[i + 1]);
    if (s1 == s2) continue;
    line t = line(p[i], p[i + 1]);
    PT inter = (t.v * l.c - l.v * t.c) /
        cross(l.v, t.v);
    T tmp = dot(inter, l.v);
    int f;
    if (s1 > s2) f = s1 && s2 ? 2 : 1;
    else f = s1 && s2 ? -2 : -1;
    vec.push_back(make_pair((f > 0 ? tmp - eps
        : tmp + eps), f)); // keep eps very
        small like 1e-12
  }
  sort(vec.begin(), vec.end());
  for (int i = 0, j = 0; i + 1 <
      (int)vec.size(); i++){
    j += vec[i].second;
    if (j) ans += vec[i + 1].first -
        vec[i].first; // if this portion is
        inside the polygon
    // else ans = 0; // if we want the maximum
        intersected length which is totally
        inside the polygon, uncomment this and
        take the maximum of ans
  }
  ans = ans / sqrt(dot(l.v, l.v));
  p.pop_back();
  return ans;
}
// given a convex polygon p, and a line ab and
    the top vertex of the polygon
// returns the intersection of the line with
    the polygon
// it returns the indices of the edges of the
    polygon that are intersected by the line
// so if it returns i, then the line intersects
    the edge (p[i], p[(i + 1) % n])
array<int, 2>
    convex_line_intersection(vector<PT> &p, PT
    a, PT b, int top) {
  int end_a = extreme_vertex(p, (a -
      b).perp(), top);
  int end_b = extreme_vertex(p, (b -
      a).perp(), top);
  auto cmp_l = [&](int i) { return
      orientation(a, p[i], b); };
  if (cmp_l(end_a) < 0 || cmp_l(end_b) > 0)
    return {-1, -1}; // no intersection
  array<int, 2> res;
  for (int i = 0; i < 2; i++) {
    int lo = end_b, hi = end_a, n = p.size();
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) /
          2) % n;
      (cmp_l(m) == cmp_l(end_b) ? lo : hi) =
          m;
    }
    res[i] = (lo + !cmp_l(hi)) % n;
    swap(end_a, end_b);
  }
  if (res[0] == res[1]) return {res[0], -1};
      // touches the vertex res[0]
  if (!cmp_l(res[0]) && !cmp_l(res[1]))
    switch ((res[0] - res[1] + (int)p.size()
        + 1) % p.size()) {
      case 0: return {res[0], res[0]}; //
          touches the edge (res[0], res[0] +
          1)
      case 2: return {res[1], res[1]}; //
          touches the edge (res[1], res[1] +
          1)
    }
  return res; // intersects the edges (res[0],
      res[0] + 1) and (res[1], res[1] + 1)
}

pair<PT, int> point_poly_tangent(vector<PT>
    &p, PT Q, int dir, int l, int r) {
  while (r - l > 1) {
    int mid = (l + r) >> 1;
    bool pvs = orientation(Q, p[mid], p[mid -
        1]) != -dir;
    bool nxt = orientation(Q, p[mid], p[mid +
        1]) != -dir;
    if (pvs && nxt) return {p[mid], mid};
    if (!(pvs || nxt)) {
      auto p1 = point_poly_tangent(p, Q, dir,
          mid + 1, r);
      auto p2 = point_poly_tangent(p, Q, dir,
          l, mid - 1);
      return orientation(Q, p1.first,
          p2.first) == dir ? p1 : p2;
    }
    if (!pvs) {
      if (orientation(Q, p[mid], p[l]) == dir)
          r = mid - 1;
      else if (orientation(Q, p[l], p[r]) ==
          dir) r = mid - 1;
      else l = mid + 1;
    }
    if (!nxt) {
      if (orientation(Q, p[mid], p[l]) == dir)
          l = mid + 1;
      else if (orientation(Q, p[l], p[r]) ==
          dir) r = mid - 1;
      else l = mid + 1;
    }
  }
  pair<PT, int> ret = {p[l], l};
  for (int i = l + 1; i <= r; i++) ret =
      orientation(Q, ret.first, p[i]) != dir ?
      make_pair(p[i], i) : ret;
  return ret;
}
// (ccw, cw) tangents from a point that is
    outside this convex polygon
// returns indexes of the points
// ccw means the tangent from Q to that point
    is in the same direction as the polygon ccw
    direction
pair<int, int>
    tangents_from_point_to_polygon(vector<PT>
    &p, PT Q){
  int ccw = point_poly_tangent(p, Q, 1, 0,
      (int)p.size() - 1).second;
  int cw = point_poly_tangent(p, Q, -1, 0,
      (int)p.size() - 1).second;
  return make_pair(ccw, cw);
}

// minimum distance from a point to a convex
    polygon
// it assumes point lie strictly outside the
    polygon
T dist_from_point_to_polygon(vector<PT> &p, PT
    z) {
  T ans = inf;
  int n = p.size();
  if (n <= 3) {
    for(int i = 0; i < n; i++) ans = min(ans,
        dist_from_point_to_seg(p[i], p[(i + 1)
        % n], z));
    return ans;
  }
  auto [r, l] =
      tangents_from_point_to_polygon(p, z);
  if(l > r) r += n;
  while (l < r) {
    int mid = (l + r) >> 1;
    T left = dist2(p[mid % n], z), right=
        dist2(p[(mid + 1) % n], z);
    ans = min({ans, left, right});
    if(left < right) r = mid;
    else l = mid + 1;
  }
  ans = sqrt(ans);
  ans = min(ans, dist_from_point_to_seg(p[l %
      n], p[(l + 1) % n], z));
  ans = min(ans, dist_from_point_to_seg(p[l %
      n], p[(l - 1 + n) % n], z));
  return ans;
}
// minimum distance from convex polygon p to
    line ab
// returns 0 is it intersects with the polygon
// top - upper right vertex
T dist_from_polygon_to_line(vector<PT> &p, PT
    a, PT b, int top) { //O(log n)
  PT orth = (b - a).perp();
  if (orientation(a, b, p[0]) > 0) orth = (a
      - b).perp();
  int id = extreme_vertex(p, orth, top);
  if (dot(p[id] - a, orth) > 0) return 0.0;
      //if orth and a are in the same half of
      the line, then poly and line intersects
  return dist_from_point_to_line(a, b,
      p[id]); //does not intersect
}
// minimum distance from a convex polygon to
    another convex polygon
// the polygon doesnot overlap or touch
T dist_from_polygon_to_polygon(vector<PT>
    &p1, vector<PT> &p2) { // O(n log n)
  T ans = inf;
  for (int i = 0; i < p1.size(); i++) {
    ans = min(ans,
        dist_from_point_to_polygon(p2,
        p1[i]));
  }
  for (int i = 0; i < p2.size(); i++) {
    ans = min(ans,
        dist_from_point_to_polygon(p1,
        p2[i]));
  }
  return ans;
}
// calculates the area of the union of n
    polygons (not necessarily convex).
// the points within each polygon must be given
    in CCW order.
// complexity: O(N^2), where N is the total
    number of points
T rat(PT a, PT b, PT p) {
    return !sign(a.x - b.x) ? (p.y - a.y) /
        (b.y - a.y) : (p.x - a.x) / (b.x -
        a.x);
 };
T polygon_union(vector<vector<PT>> &p) {
  int n = p.size();
  T ans=0;
  for(int i = 0; i < n; ++i) {
    for (int v = 0; v < (int)p[i].size(); ++v)
        {
      PT a = p[i][v], b = p[i][(v + 1) %
          p[i].size()];
      vector<pair<T, int>> segs;
      segs.emplace_back(0,  0),
          segs.emplace_back(1,  0);
      for(int j = 0; j < n; ++j) {
        if(i != j) {
```

```
        for(size_t u = 0; u < p[j].size();
            ++u) {
          PT c = p[j][u], d = p[j][(u + 1) %
              p[j].size()];
          int sc = sign(cross(b - a, c -
              a)), sd = sign(cross(b - a, d
              - a));
          if(!sc && !sd) {
            if(sign(dot(b - a, d - c)) > 0
                && i > j) {
              segs.emplace_back(rat(a, b,
                  c), 1),
                  segs.emplace_back(rat(a,
                  b, d),  -1);
            }
          }
          else {
            T sa = cross(d - c, a - c), sb
                = cross(d - c, b - c);
            if(sc >= 0 && sd < 0)
                segs.emplace_back(sa / (sa
                - sb), 1);
            else if(sc < 0 && sd >= 0)
                segs.emplace_back(sa / (sa
                - sb),  -1);
          }
        }
      }
    }
    sort(segs.begin(),  segs.end());
    T pre = min(max(segs[0].first, 0.0),
        1.0), now, sum = 0;
    int cnt = segs[0].second;
    for(int j = 1; j < segs.size(); ++j) {
      now = min(max(segs[j].first, 0.0),
          1.0);
      if (!cnt) sum += now - pre;
      cnt += segs[j].second;
      pre = now;
    }
    ans += cross(a, b) * sum;
  }
 }
 return ans * 0.5;
}
// returns the area of the intersection of the
    circle with center c and radius r
// and the triangle formed by the points c, a,
    b
T _triangle_circle_intersection(PT c, T r, PT
    a, PT b) {
 T sd1 = dist2(c, a), sd2 = dist2(c, b);
 if(sd1 > sd2) swap(a, b), swap(sd1, sd2);
 T sd = dist2(a, b);
 T d1 = sqrtl(sd1), d2 = sqrtl(sd2), d =
     sqrt(sd);
 T x = abs(sd2 - sd - sd1) / (2 * d);
 T h = sqrtl(sd1 - x * x);
 if(r >= d2) return h * d / 2;
 T area = 0;
 if(sd + sd1 < sd2) {
   if(r < d1) area = r * r * (acos(h / d2) -
       acos(h / d1)) / 2;
   else {
     area = r * r * ( acos(h / d2) - acos(h /
         r)) / 2;
     T y = sqrtl(r * r - h * h);
     area += h * (y - x) / 2;
   }
 }
 else {
   if(r < h) area = r * r * (acos(h / d2) +
       acos(h / d1)) / 2;
   else {
     area += r * r * (acos(h / d2) - acos(h /
         r)) / 2;
     T y = sqrtl(r * r - h * h);
     area += h * y / 2;
     if(r < d1) {
       area += r * r * (acos(h / d1) - acos(h
           / r)) / 2;
       area += h * y / 2;
     }
     else area += h * x / 2;
   }
 }
 return area;
}
// Closest-Pair of Points (O(n log n))
// Returns minimal distance among all pairs in
    v
T closest(vector<PT>& v){
 sort(v.begin(), v.end(), [](PT a, PT b){
     return a.x < b.x; });
 function<T(int,int)> rec = [&](int l,int r){
   if(r-l < 2) return
       numeric_limits<T>::infinity();
   int m = (l+r)/2;
   T d = min(rec(l,m), rec(m,r));
   inplace_merge(v.begin()+l, v.begin()+m,
       v.begin()+r, [](PT a, PT b){ return
       a.y < b.y; });
   vector<PT> buf;
   for(int i=l; i<r; i++){
     if(fabs(v[i].x - v[m].x) < d){
       for(int j=(int)buf.size()-1; j>=0 &&
           v[i].y - buf[j].y < d; --j)
         d = min(d, dist(v[i], buf[j]));
       buf.push_back(v[i]);
     }
   }
   return d;
 };
 return rec(0, v.size());
}
```

6.3 Minkowski Sum [44 lines]

```
struct pt{
    long long x, y;
    pt operator + (const pt & p) const {
        return pt{x + p.x, y + p.y};
    }
    pt operator - (const pt & p) const {
        return pt{x - p.x, y - p.y};
    }
    long long cross(const pt & p) const {
        return x * p.y - y * p.x;
    }
};

void reorder_polygon(vector<pt> & P){
    size_t pos = 0;
    for(size_t i = 1; i < P.size(); i++){
        if(P[i].y < P[pos].y || (P[i].y ==
            P[pos].y && P[i].x < P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos,
        P.end());
}

vector<pt> minkowski(vector<pt> P, vector<pt>
    Q){
    // the first vertex must be the lowest
    reorder_polygon(P);
    reorder_polygon(Q);
    // we must ensure cyclic indexing
    P.push_back(P[0]);
    P.push_back(P[1]);
    Q.push_back(Q[0]);
    Q.push_back(Q[1]);
    // main part
    vector<pt> result;
    size_t i = 0, j = 0;
    while(i < P.size() - 2 || j < Q.size() -
        2){
        result.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] -
            P[i]).cross(Q[j + 1] - Q[j]);
        if(cross >= 0 && i < P.size() - 2)
            ++i;
        if(cross <= 0 && j < Q.size() - 2)
            ++j;
    }
    return result;
}
```

7 Graph

7.1 Articulation point and bridge [22 lines]

```
const int N=1e5+5;
vector<int> g[N];
int vis[N],dis[N],lo[N],isAP[N];
int timer;
vector<pair<int,int>> ans;
void dfs(int src, int par)
{
    int child = 0;
    vis[src] = true;
    dis[src] = lo[src] = ++timer;
    for (auto v : g[src]) {
        if (!vis[v]) {
            child++;
            dfs(v,src);
            lo[src] = min(lo[src], lo[v]);
            if (par != -1 && lo[v] >=
                dis[src]) isAP[src] = true;
            if(lo[v]>dis[src])
                ans.push_back({min(v,src),
                max(v,src)});
        }
        else if (v != par) lo[src] =
            min(lo[src], dis[v]);
    }
    if (par == -1 && child > 1) isAP[src] =
        true;
}
```

7.2 Bellman Ford [21 lines]

```
void bellman()
{
    for(int i=0;i<n;i++)
    {
        for(int node=1;node<=n;node++)
        {
            for(auto adj_node:adj_list[node])
            {
                int u=node;
                int v=adj_node.first;
                int w=adj_node.second;
                if(d[v]>d[u]+w)
                {
                    d[v]=d[u]+w;
                    parent[v]=u;
                    if(i==n-1)
                        negative_cycle=true,
                        selected_node=v;
                }
            }
        }
    }
}
```

7.3 Floyd Warshall [6 lines]

```
void floyd()
{
    for(int k=1;k<=n;k++)
        for(int u=1;u<=n;u++)
            for(int v=1;v<=n;v++)
                d[u][v]=min(d[u][k]+d[k][v],
                d[u][v]);
}
```

7.4 SCC [59 lines]

```
vector<vector<int>>g,rg;
vector<bool>vis;
int timer;
int stime[N];
int etime[N];
void dfs(int node)
{
    if(vis[node])return;
    vis[node]=1;
    stime[node]=++timer;
    for(auto it:g[node])
    {
        dfs(it);
    }
    etime[node]=++timer;
}
void rdfs(int node,vector<int>&tem)
{
    if(vis[node])return;
    vis[node]=1;
    tem.push_back(node);
    for(auto it:rg[node])
    {
        rdfs(it,tem);
    }
}
//main
int main()
{
```

```
    int n,m;
    cin>>n>>m;
    g.assign(n+1,vector<int>(0));
    rg.assign(n+1,vector<int>(0));
    vis.assign(n+1,0);
    timer=0;
    for(int i=0;i<m;i++)
    {
        int x,y;
        cin>>x>>y;
        g[x].push_back(y);
        rg[y].push_back(x);
    }
    vector<pair<int,int>>str;
    for(int i=1;i<=n;i++)
    {
        dfs(i);
        str.push_back({etime[i],i});
    }
    sort(all(str));
    reverse(all(str));
    vis.assign(n+1,0);
    for(auto [ww,i]:str)
    {
        if(vis[i])continue;
        vector<int>tem;
        rdfs(i,tem);
        for(auto it:tem)cout<<it<<" ";
            cout<<"\n";
    }
}
```

7.5 TopoSort [26 lines]

```
vector<vector<int>>g;
vector<int>toposort(int n)
{
    queue<int>str;
    vector<int>in(n+1,0);
    for(int i=1;i<=n;i++)
    {
        for(auto it:g[i])in[it]++;
    }
    for(int
        i=1;i<=n;i++)if(in[i]==0)str.push(i);
    vector<int>ans;
    while(!str.empty())
    {
        auto x=str.front();
        str.pop();
        ans.push_back(x);
        for(auto it:g[x])
        {
            in[it]--;
            if(in[it]==0)str.push(it);
        }
    }
    return ans;
}
//g.resize(n+1,vector<int>(0));
//if(ans.size()!=n)cout<<"-1\n";
```

8 Math

8.1 Berlekamp massey [180 lines]

```
#include <bits/stdc++.h>
using namespace std;
using int64 = long long;
const int64 MOD = 1000000007;

int64 modpow(int64 a, int64 e){
    int64 r=1%MOD;
    a%=MOD;
    while(e){
        if(e&1) r = (__int128)r*a % MOD;
        a = (__int128)a*a % MOD;
        e >>= 1;
    }
    return r;
}
int64 modinv(int64 x){ return
    modpow((x%MOD+MOD)%MOD, MOD-2); }

// --- Fast doubling for Fibonacci (returns
    pair (F_n, F_{n+1})) ---
pair<int64,int64> fib_pair(long long n){
    if(n==0) return {0,1};
    auto p = fib_pair(n>>1);
    int64 a = p.first, b = p.second;
    int64 c = ( (__int128)a * ( (2*b - a +
        MOD) % MOD ) ) % MOD; // F(2k)
    int64 d = ( (__int128)a*a + (__int128)b*b
        ) % MOD;            // F(2k+1)
    if(n&1) return {d, (c+d)%MOD};
    else    return {c,d};
}
int64 fib(long long n){ return
    fib_pair(n).first; }

// --- Matrix utilities for small matrices
    (square) ---
using Mat = vector<vector<int64>>;

Mat matMul(const Mat &A, const Mat &B){
    int n = A.size();
    int m = B[0].size();
    int p = B.size();
    Mat C(n, vector<int64>(m,0));
    for(int i=0;i<n;i++){
        for(int k=0;k<p;k++){
            if(A[i][k]==0) continue;
            int64 av = A[i][k];
            for(int j=0;j<m;j++){
                C[i][j] = (C[i][j] + av *
                    B[k][j]) % MOD;
            }
        }
    }
    return C;
}
Mat matPow(Mat base, long long e){
    int n = base.size();
    Mat R(n, vector<int64>(n,0));
    for(int i=0;i<n;i++) R[i][i]=1;
    while(e){
        if(e&1) R = matMul(R, base);
        base = matMul(base, base);
        e >>= 1;
    }
    return R;
}

// --- Berlekamp-Massey: returns coefficients c
    (length L) so that
//     a_n = c[0]*a_{n-1} + c[1]*a_{n-2} + ...
    + c[L-1]*a_{n-L}
//     (i.e., recurrence of order L). Works
    modulo MOD. ---
vector<int64> berlekamp_massey(const
    vector<int64>& s){
    int n = s.size();
    vector<int64> C(1,1), B(1,1);
    int L = 0, m = 1;
    int64 b = 1;
    for(int i=0;i<n;i++){
        // compute discrepancy
        int64 d = 0;
        for(int j=0;j<=L;j++){
            d = (d + C[j] * s[i-j]) % MOD;
        }
        if(d==0){
            m++;
        } else {
            vector<int64> T = C;
            int64 coef = d * modinv(b) % MOD;
            // C = C - coef * x^m * B
            int need = max((int)C.size(),
                (int)B.size() + m);
            C.resize(need);
            for(size_t j=0;j<B.size();j++){
                int idx = j + m;
                C[idx] = (C[idx] - coef *
                    B[j]) % MOD;
                if(C[idx]<0) C[idx]+=MOD;
            }
            if(2*L <= i){
                L = i+1 - L;
                B = T;
                b = d;
                m = 1;
            } else {
                m++;
            }
        }
    }
    // currently C represents polynomial with
        leading 1 and C[0]=1
    // We want recurrence a_n = sum_{i=1..L}
        coeff[i-1] * a_{n-i}
    // For that, coeff[i-1] = (MOD - C[i]) %
        MOD  for i=1..L
    vector<int64> res;
    for(int i=1;i<(int)C.size();i++){
        res.push_back( (MOD - C[i]) % MOD );
    }
    return res;
}

// --- Helper: compute sum_{i=0..N} a_i given
    recurrence and initial values ---
// recurrence length L, coeffs c[0..L-1],
    initial a[0..L-1]
int64 sum_linear_recurrence(const
    vector<int64>& c, const vector<int64>&
    init, long long N){
    int L = c.size();
    if(N < (int)init.size()){
        int64 s=0;
        for(int i=0;i<=N;i++) s = (s +
            init[i]) % MOD;
        return s;
    }
    // Build augmented matrix M of size
        (L+1)x(L+1)
    int sz = L+1;
    Mat M(sz, vector<int64>(sz,0));
    // top row -> next value a_{n+1} = sum
        c[j]*a_{n-j}
    for(int j=0;j<L;j++) M[0][j] = c[j] % MOD;
    // shift rows
    for(int i=1;i<L;i++){
        M[i][i-1] = 1;
    }
    // last row: S_{n+1} = S_n + a_{n+1} = S_n
        + dot(c, top L entries)
    for(int j=0;j<L;j++) M[L][j] = c[j] % MOD;
    M[L][L] = 1;
    // initial state vector at index n = L-1:
    // u_{L-1} = [ a_{L-1}, a_{L-2}, ..., a_0,
        S_{L-1} ]^T
    Mat u(sz, vector<int64>(1,0));
    int64 S_init = 0;
    for(int i=0;i<L;i++) S_init = (S_init +
        init[i]) % MOD;
    for(int i=0;i<L;i++){
        u[i][0] = init[L-1 - i]; // top
            element a_{L-1}
    }
    u[L][0] = S_init;
    long long exp = N - (L - 1);
    Mat P = matPow(M, exp);
    Mat res = matMul(P, u);
    int64 ans = res[L][0] % MOD; if(ans<0)
        ans+=MOD;
    return ans;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int T;
    if(!(cin>>T)) return 0;
    for(int tc=1; tc<=T; ++tc){
        long long N; long long C; int K;
        cin >> N >> C >> K;
        // We need sequence a_n = (F_{n*C})^K
            for n=0..N
        // Prepare initial terms: need at least
            2*(K+1) terms to run BM safely.
        int need = 2*(K+1) + 5; // a bit extra
            margin
        vector<int64> seq;
        seq.reserve(need);
        for(int i=0;i<need;i++){
            int64 f = fib((long long)i * C);
                // F_{iC} mod MOD
            int64 val = modpow(f, K);
            seq.push_back(val);
        }
        // Run Berlekamp-Massey on seq to get
            recurrence
        vector<int64> rec =
            berlekamp_massey(seq);
```

```
        int L = rec.size();
        if(L==0){
            // sequence is all zero -> sum=0
            cout << "Case " << tc << ": 0\n";
            continue;
        }
        // initial a[0..L-1]
        vector<int64> init(L);
        for(int i=0;i<L;i++) init[i] = seq[i]
            % MOD;
        // Compute sum from 0..N
        int64 ans =
            sum_linear_recurrence(rec, init,
            N);
        cout << "Case " << tc << ": " << ans %
            MOD << "\n";
    }
    return 0;
}
```

8.2 Convolution [192 lines]

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using i128 = __int128_t;

ll modpow(ll a, ll e, ll mod)
{
    ll r = 1;
    a %= mod;
    while (e)
    {
        if (e & 1) r = (i128)r * a % mod;
        a = (i128)a * a % mod;
        e >>= 1;
    }
    return r;
}
ll modinv(ll a, ll mod) { return modpow((a %
    mod + mod) % mod, mod - 2, mod); }

struct XOR
{
    static inline void fwd(ll &u, ll &v)
    {
        ll x = u + v, y = u - v;
        u = x;
        v = y;
    }
    static inline void inv(ll &u, ll &v)
    {
        ll x = u + v, y = u - v;
        u = x;
        v = y;
    }
};
struct OR
{
    static inline void fwd(ll &u, ll &v) { v
        += u; }
    static inline void inv(ll &u, ll &v) { v
        -= u; }
};
struct AND
{
    static inline void fwd(ll &u, ll &v) { u
        += v; }
    static inline void inv(ll &u, ll &v) { u
        -= v; }
};

template <class Op>
void fwht(vector<ll> &a, bool invflag = false)
{
    int n = (int)a.size();
    for (int len = 1; len < n; len <<= 1)
    {
        for (int i = 0; i < n; i += len << 1)
        {
            for (int j = 0; j < len; ++j)
            {
                ll &x = a[i + j], &y = a[i + j
                    + len];
                if (!invflag) Op::fwd(x, y);
                else Op::inv(x, y);
            }
        }
    }
    if (invflag && is_same<Op, XOR>::value)
        for (int i = 0; i < n; ++i) a[i] /= n;
}

template <class Op>
void fwht_mod(vector<ll> &a, ll mod, bool
    invflag = false)
{
    int n = (int)a.size();
    for (int len = 1; len < n; len <<= 1)
    {
        for (int i = 0; i < n; i += len << 1)
        {
            for (int j = 0; j < len; ++j)
            {
                ll x = a[i + j], y = a[i + j
                    + len];
                if (!invflag)
                {
                    if constexpr (is_same<Op,
                        XOR>::value)
                    {
                        ll nx = (x + y) %
                            mod, ny = (x - y)
                            % mod;
                        if (ny < 0) ny += mod;
                        a[i + j] = nx;
                        a[i + j + len] = ny;
                    }
                    else if constexpr
                        (is_same<Op,
                        OR>::value)
                        a[i + j + len] = (y +
                            x) % mod;
                    else a[i + j] = (x + y) %
                        mod;
                }
                else
                {
                    if constexpr (is_same<Op,
                        XOR>::value)
                    {
                        ll nx = (x + y) %
                            mod, ny = (x - y)
                            % mod;
                        if (ny < 0) ny += mod;
                        a[i + j] = nx;
                        a[i + j + len] = ny;
                    }
                    else if constexpr
                        (is_same<Op,
                        OR>::value)
                    {
                        ll ny = (y - x) % mod;
                        if (ny < 0) ny += mod;
                        a[i + j + len] = ny;
                    }
                    else
                    {
                        ll nx = (x - y) % mod;
                        if (nx < 0) nx += mod;
                        a[i + j] = nx;
                    }
                }
            }
        }
    }
    if (invflag && is_same<Op, XOR>::value)
    {
        ll invn = modinv(n, mod);
        for (int i = 0; i < n; ++i)
            a[i] = (a[i] * (i128)invn) % mod;
    }
}

template <class Op>
vector<ll> conv(vector<ll> a, vector<ll> b)
{
    int n = 1;
    while (n < (int)max(a.size(), b.size())) n
        <<= 1;
    a.resize(n);
    b.resize(n);
    fwht<Op>(a, false);
    fwht<Op>(b, false);
    for (int i = 0; i < n; ++i) a[i] = a[i] *
        b[i];
    fwht<Op>(a, true);
    return a;
}


template <class Op>
vector<ll> conv_mod(vector<ll> a, vector<ll>
    b, ll mod)
{
    int n = 1;
    while (n < (int)max(a.size(), b.size())) n
        <<= 1;
    a.resize(n);
    b.resize(n);
    fwht_mod<Op>(a, mod, false);
    fwht_mod<Op>(b, mod, false);
    vector<ll> c(n);
    for (int i = 0; i < n; ++i)
        c[i] = (i128)a[i] * b[i] % mod;
    fwht_mod<Op>(c, mod, true);
    return c;
}

// quick usage demo
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // XOR integer conv
    {
        vector<ll> A = {1, 2, 3, 4}, B = {5,
            6, 7, 8};
        auto C = conv<XOR>(A, B);
        for (auto x : C)
            cout << x << ' ';
        cout << "\n";
    }

    // OR integer conv (size must be pow2)
    {
        vector<ll> A(8), B(8);
        A[0] = 1;
        A[1] = 2;
        B[0] = 3;
        B[4] = 1;
        auto C = conv<OR>(A, B);
        for (auto x : C)
            cout << x << ' ';
        cout << "\n";
    }

    // XOR modular conv
    {
        ll MOD = 1000000007;
        vector<ll> A = {1, 2, 3, 4}, B = {5,
            6, 7, 8};
        auto C = conv_mod<XOR>(A, B, MOD);
        for (auto x : C)
            cout << x << ' ';
        cout << "\n";
    }
    return 0;
}
```

8.3 FFT [73 lines]

```
#include<bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;
const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a),
      b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b
        + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>=
        1);
```

```
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<=
      1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.)
        * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j
          < k; ++j, w = w * wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a
      /= n, p[i].b /= n;
}
vector<long long> multiply(vector<int> &a,
    vector<int> &b) {
  int n = a.size(), m = b.size(), t = n + m -
      1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) :
        base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) :
        base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] *
      y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long
      long) round(z[i].a);
  while((int)ret.size() > 1 && ret.back() ==
      0) ret.pop_back();
  return ret;
}
long long ans[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, x; cin >> n >> x;
  vector<int> a(n + 1, 0), b(n + 1, 0), c(n +
      1, 0);
  int nw = 0;
  a[0]++; b[n]++;
  long long z = 0;
  for (int i = 1; i <= n; i++) {
    int k; cin >> k;
    nw += k < x;
    a[nw]++; b[-nw + n]++;
    z += c[nw] + !nw; c[nw]++;
  }
  auto res = multiply(a, b);
  for (int i = n + 1; i < res.size(); i++) {
    ans[i - n] += res[i];
  }
  ans[0] = z;
  for (int i = 0; i <= n; i++) cout << ans[i]
      << ' ';
  cout << '\n';
  return 0;
}
```

8.4 NTT [65 lines]

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1 << 20;
const int mod = 998244353;
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
int POW(int x, int y, int ans = 1) {
  for (; y; y >>= 1, x = (long long) x * x %
      mod) if (y & 1) ans = (long long) ans *
      x % mod;
  return ans;
}
void precompute(int len) {
  lim = wn[0] = 1; int s = -1;
  while (lim < len) lim <<= 1, ++s;
  for (int i = 0; i < lim; ++i) rev[i] = rev[i
      >> 1] >> 1 | (i & 1) << s;
  const int g = POW(root, (mod - 1) / lim);
  inv_lim = POW(lim, mod - 2);
  for (int i = 1; i < lim; ++i) wn[i] = (long
      long) wn[i - 1] * g % mod;
}
void ntt(vector<int> &a, int typ) {
  for (int i = 0; i < lim; ++i) if (i <
      rev[i]) swap(a[i], a[rev[i]]);
  for (int i = 1; i < lim; i <<= 1) {
    for (int j = 0, t = lim / i / 2; j < i;
        ++j) w[j] = wn[j * t];
    for (int j = 0; j < lim; j += i << 1) {
      for (int k = 0; k < i; ++k) {
        const int x = a[k + j], y = (long
            long) a[k + j + i] * w[k] % mod;
        reduce(a[k + j] += y - mod),
            reduce(a[k + j + i] = x - y);
      }
    }
  }
  if (!typ) {
    reverse(a.begin() + 1, a.begin() + lim);
    for (int i = 0; i < lim; ++i) a[i] = (long
        long) a[i] * inv_lim % mod;
  }
}
vector<int> multiply(vector<int> &f,
    vector<int> &g) {
  if (f.empty() or g.empty()) return {};
  int n = (int)f.size() + (int)g.size() - 1;
  if (n == 1) return {(int)((long long) f[0]
      * g[0] % mod)};
  precompute(n);
  vector<int> a = f, b = g;
  a.resize(lim); b.resize(lim);
  ntt(a, 1), ntt(b, 1);
  for (int i = 0; i < lim; ++i) a[i] = (long
      long) a[i] * b[i] % mod;
  ntt(a, 0);
  a.resize(n + 1);
  return a;
}
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  vector<int> a(n), b(m);
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  for (int i = 0; i < m; i++) {
    cin >> b[i];
  }
  auto ans = multiply(a, b);
  ans.resize(n + m - 1);
  for (auto x: ans) cout << x << ' '; cout <<
      '\n';
  return 0;
}
```

8.5 NTT_with_Any_prime_MOD [125 lines]

```
#include<bits/stdc++.h>
using namespace std;

const int N = 3e5 + 9, mod = 998244353;

struct base {
  double x, y;
  base() { x = y = 0; }
  base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) {
    return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) {
    return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) {
    return base(a.x * b.x - a.y * b.y, a.x *
    b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x,
    -a.y); }
int lim = 1;
vector<base> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};
const double PI = acosl(- 1.0);
void ensure_base(int p) {
  if(p <= lim) return;
  rev.resize(1 << p);
  for(int i = 0; i < (1 << p); i++) rev[i] =
      (rev[i >> 1] >> 1) + ((i & 1)  <<  (p -
      1));
  roots.resize(1 << p);
  while(lim < p) {
    double angle = 2 * PI / (1 << (lim + 1));
    for(int i = 1 << (lim - 1); i < (1 <<
        lim); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1
          << lim));
      roots[(i << 1) + 1] =
          base(cos(angle_i), sin(angle_i));
    }
    lim++;
  }
}
void fft(vector<base> &a, int n = -1) {
  if(n == -1) n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = lim - zeros;
  for(int i = 0; i < n; i++) if(i < (rev[i]
      >> shift)) swap(a[i], a[rev[i] >>
      shift]);
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        base z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a,
    vector<int> &b, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int p = 0;
  while((1 << p) < need) p++;
  ensure_base(p);
  int sz = 1 << p;
  vector<base> A, B;
  if(sz > (int)A.size()) A.resize(sz);
  for(int i = 0; i < (int)a.size(); i++) {
    int x = (a[i] % mod + mod) % mod;
    A[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(A.begin() + a.size(), A.begin() + sz,
      base{0, 0});
  fft(A, sz);
  if(sz > (int)B.size()) B.resize(sz);
  if(eq) copy(A.begin(), A.begin() + sz,
      B.begin());
  else {
    for(int i = 0; i < (int)b.size(); i++) {
      int x = (b[i] % mod + mod) % mod;
      B[i] = base(x & ((1 << 15) - 1), x >>
          15);
    }
    fill(B.begin() + b.size(), B.begin() +
        sz, base{0, 0});
    fft(B, sz);
  }
  double ratio = 0.25 / sz;
  base r2(0,  - 1), r3(ratio, 0), r4(0,  -
      ratio), r5(0, 1);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    base a1 = (A[i] + conj(A[j])), a2 = (A[i]
        - conj(A[j])) * r2;
    base b1 = (B[i] + conj(B[j])) * r3, b2 =
        (B[i] - conj(B[j])) * r4;
    if(i != j) {
      base c1 = (A[j] + conj(A[i])), c2 =
          (A[j] - conj(A[i])) * r2;
      base d1 = (B[j] + conj(B[i])) * r3, d2
          = (B[j] - conj(B[i])) * r4;
      A[i] = c1 * d1 + c2 * d2 * r5;
      B[i] = c1 * d2 + c2 * d1;
    }
    A[j] = a1 * b1 + a2 * b2 * r5;
    B[j] = a1 * b2 + a2 * b1;
```

```
  }
  fft(A, sz); fft(B, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    long long aa = A[i].x + 0.5;
    long long bb = B[i].x + 0.5;
    long long cc = A[i].y + 0.5;
    res[i] = (aa + ((bb % mod) << 15) + ((cc %
        mod) << 30))%mod;
  }
  return res;
}

vector<int> pow(vector<int>& a, int p) {
  vector<int> res;
  res.emplace_back(1);
  while(p) {
    if(p & 1) res = multiply(res, a);
    a = multiply(a, a, 1);
    p >>= 1;
  }
  return res;
}
int main() {
  int n, k; cin >> n >> k;
  vector<int> a(10, 0);
  while(k--) {
    int m; cin >> m;
    a[m] = 1;
  }
  vector<int> ans = pow(a, n / 2);
  int res = 0;
  for(auto x: ans) res = (res + 1LL * x * x %
      mod) % mod;
  cout << res << '\n';
  return 0;
}
```

9 Matrix

9.1 Inverse Matrix [66 lines]

```
vector<vector<int>>
    find_cofactor(vector<vector<int>> a, int
    r, int c)
{
    vector<vector<int>> mat;
    int n = a.size();
    for (int i = 0; i < n; i++)
    {
        vector<int> row;
        if (r == i) continue;
        for (int j = 0; j < n; j++)
        {
            if (j == c) continue;
            row.push_back(a[i][j]);
        }
        mat.push_back(row);
    }
    return mat;
}
int determinant(vector<vector<int>> a)
{
    if (a.size() == 1) return a[0][0];
    int n = a.size();
    int sign = +1;
    int det = 0;
    for (int i = 0; i < n; i++)
    {
        vector<vector<int>> cf_mat =
            find_cofactor(a, 0, i);
        int cofactor = determinant(cf_mat);
        det += cofactor * sign * a[0][i];
        sign = -sign;
    }
    return det;
}
vector<vector<int>>
    transpose(vector<vector<int>> a)
{
    int n = a.size();
    vector<vector<int>> res(n,
        vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            res[i][j] = a[j][i];
    return res;
}
vector<vector<double>>
    find_inverse(vector<vector<int>> &a)
{
    int n = a.size();
    int det = determinant(a);
    if (det == 0)
    {
        cout << "Inverse Impossible\n";
        return {{}};
    }
    vector<vector<int>>
        cofactor_matrix(n,vector<int>(n));
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            int sign = (i + j) % 2 ? -1 : +1;
            cofactor_matrix[i][j] = sign
                *determinant(find_cofactor(a,
                i, j));
        }
    }
    auto adj_matrix =
        transpose(cofactor_matrix);
    auto inverse_mat = vector<vector<double>>
        (n,vector<double>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            inverse_mat[i][j] = 1.0 *
                adj_matrix[i][j] /det;
    return inverse_mat;
}
```

9.2 Matrix Multiplication [43 lines]

```
const ll M=1e9+7;
const int N=103;
int m;
ll mat[N][N];
ll ans[N][N];
void pow(int po)
{
    for(int i=0; i<m; i++)
        for(int j=0; j<m;
            j++)ans[i][j]=(i==j);
    while(po)
    {
        if(po%2)
        {
            ll tem[m][m];
            for(int i=0; i<m; i++)
            {
                for(int j=0; j<m; j++)
                {
                    tem[i][j]=0;
                    for(int k=0; k<m; k++)
                        tem[i][j]=(tmp[i][j]+
                            mat[i][k]*
                            ans[k][j])%M;
                }
            }
            for(int i=0; i<m; i++)
                for(int j=0; j<m; j++)
                    ans[i][j]=tem[i][j];
        }
        po=po/2;
        ll tem[m][m];
        for(int i=0; i<m; i++)
        {
            for(int j=0; j<m; j++)
            {
                tem[i][j]=0;
                for(int k=0; k<m; k++)
                    tem[i][j]=(tmp[i][j]+
                        mat[i][k]*
                        mat[k][j])%M;
            }
        }
        for(int i=0; i<m; i++)
            for(int j=0; j<m; j++)
                mat[i][j]=tem[i][j];
    }
}
```

9.3 Matrix [126 lines]

```
#include<bits/stdc++.h>
using namespace std;
//must 0 based
template<typename T = double>
struct Matrix {
    int n, m;
    vector<vector<T>> a;
    Matrix(): n(0), m(0) {}
    Matrix(int n_, int m_, T init = T()):
        n(n_), m(m_), a(n_, vector<T>(m_,
        init)) {}
    vector<T>& operator[](int i){ return
        a[i]; }
    const vector<T>& operator[](int i) const {
        return a[i]; }
    static Matrix Identity(int k) {
        Matrix I(k,k,T());
        for(int i=0;i<k;i++) I.a[i][i] = T(1);
        return I;
    }
    Matrix operator+(Matrix& o){
        assert(n==o.n && m==o.m);
        Matrix r(n,m);
        for(int i=0;i<n;i++) for(int
            j=0;j<m;j++) r.a[i][j] = a[i][j]
            + o.a[i][j];
        return r;
    }
    Matrix operator-(Matrix& o){
        assert(n==o.n && m==o.m);
        Matrix r(n,m);
        for(int i=0;i<n;i++) for(int
            j=0;j<m;j++) r.a[i][j] = a[i][j]
            - o.a[i][j];
        return r;
    }
    Matrix operator*(Matrix& o){
        assert(m == o.n);
        Matrix r(n, o.m, T());
        for(int i=0;i<n;i++){
            for(int k=0;k<m;k++){
                T aik = a[i][k];
                for(int j=0;j<o.m;j++){
                    r.a[i][j] += aik *
                        o.a[k][j];
                }
            }
        }
        return r;
    }
    Matrix pow(long long e){
        assert(n==m);
        if (e == 0) return Identity(n);
        if (e < 0) return inverse().pow(-e);
        Matrix base = *this, res =
            Identity(n);
        while(e){
            if (e & 1) res = res * base;
            base = base * base;
            e >>= 1;
        }
        return res;
    }
    long double determinant(double eps =
        1e-12) const {
        assert(n==m);
        int N = n;
        vector<vector<long double>> b(N,
            vector<long double>(N));
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                b[i][j]=a[i][j];
        long double det = 1.0L;
        for (int col = 0; col < N; ++col) {
            int pivot = col;
            for (int i = col + 1; i < N; ++i)
                if (fabsl(b[i][col]) >
                fabsl(b[pivot][col])) pivot =
                i;
            if (fabsl(b[pivot][col]) < eps)
                return 0; // singular
            if (pivot != col) {
                swap(b[pivot], b[col]);
                det = -det;
            }
            det *= b[col][col];
            long double inv_pivot = 1.0L /
                b[col][col];
            for (int i = col + 1; i < N; ++i)
                {
                long double factor = b[i][col]
                    * inv_pivot;
```

```
            if (fabsl(factor) < 1e-18L)
                continue;
            for (int j = col; j < N;
                ++j)b[i][j] -= factor *
                b[col][j];
        }
    }
    return det;
}
Matrix inverse(double eps = 1e-12){
    if (n != m) throw
        runtime_error("inverse requires
        square matrix");
    int N = n;
    vector<vector<double>> aug(N,
        vector<double>(2*N));
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++) aug[i][j] =
            double(a[i][j]);
        for(int j=0;j<N;j++) aug[i][N+j]
            = (i==j) ? 1.0 : 0.0;
    }
    for(int col=0; col<N; ++col){
        int pivot = col;
        for(int i=col+1;i<N;i++) if
            (fabs(aug[i][col]) >
            fabs(aug[pivot][col])) pivot
            = i;
        if (fabs(aug[pivot][col]) < eps)
            throw runtime_error("singular
            matrix (or near-singular)");
        swap(aug[col], aug[pivot]);
        double div = aug[col][col];
        for(int j=0;j<2*N;j++) aug[col][j]
            /= div;
        for(int i=0;i<N;i++){
            if (i==col) continue;
            double factor = aug[i][col];
            if (fabs(factor) < 1e-18)
                continue;
            for(int j=col;j<2*N;j++)
                aug[i][j] -= factor *
                aug[col][j];
        }
    }
    Matrix res(N,N);
    for(int i=0;i<N;i++) for(int
        j=0;j<N;j++) res.a[i][j] =
        T(aug[i][N+j]);
    return res;
}
void print(int precision = 10) {
    ios::fmtflags f = cout.flags();
    cout.setf(ios::fixed);
        cout<<setprecision(precision);
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if (j) cout<<' ';
            cout<<a[i][j];
        }
        cout<<"\n";
    }
    cout.flags(f);
}
};
```

```
int main()
{
    int n,m;
    cin >> n >> m;
    Matrix mat(n,m);
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++) cin >> mat[i][j];
    mat.print();
}
```

10 Misc

10.1 Bit hacks [25 lines]

```
# x & -x is the least bit in x.
# iterate over all the subsets of the mask
for (int s=m; ; s=(s-1)&m) {
 ... you can use s ...
 if (s==0)  break;
}
# c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is
    the
next number after x with the same number of
    bits set.
# __builtin_popcount(x) //number of ones in
    binary
  __builtin_popcountll(x) // for long long
# __builtin_clz(x) // number of leading zeros
  __builtin_ctz(x) // number of trailing
      zeros, they also have long long version
# Some properties of bitwise operations:
a|b = a xor b + a&b
a xor (a&b) = (a|b) xor b
b xor (a&b) = (a|b) xor a
(a&b) xor (a|b) = a xor b
# Addition:
a+b = a|b + a&b
a+b = a xor b + 2(a&b)
# Subtraction:
a-b = (a xor (a&b))-((a|b) xor a)
a-b = ((a|b) xor b)-((a|b) xor a)
a-b = (a xor (a&b))-(b xor (a&b))
a-b = ((a|b) xor b)-(b xor (a&b))
```

10.2 Bitset C++ [18 lines]

```
bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_first()<<endl; // prints 1
bs._Find_next(idx). This function returns
    first set bit after index idx.for example:
bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_next(1)<<','<<BS._Find_next(3)
    <<endl; // prints 7,7
So this code will print all of the set bits of
    BS:

for(int i=BS._Find_first();i< BS.size();i =
    BS._Find_next(i))
    cout<<i<<endl;
//Note that there isn't any set bit after idx,
    BS._Find_next(idx) will return BS.size();
    same as calling BS._Find_first() when
    bitset is clear;
b[i], b.test(i)
b.set(), b.set(i), b.set(i, val)
b.reset(), b.reset(i)
b.flip(), b.flip(i)
b.count(), b.size(), b.any(), b.none(),
    b.all()
b.to_ulong(), b.to_ullong(), b.to_string()
```

10.3 Template [33 lines]

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
template <typename T>using orderedSet =
    tree<T, null_type, less_equal<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;
//order_of_key(k) - number of element strictly
    less than k
//find_by_order(k) - k'th element in set.(0
    indexed)(iterator)
mt19937_64 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
long long get_rand(long long l, long long r) {
    // random number from l to r
  assert(l <= r);
  return l + rnd() % (r - l + 1);
}
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const {
    static const uint64_t FIXED_RANDOM =
        chrono::steady_clock::now().
        time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};
//pair (a, b) er jonne a * MOD + b
gp_hash_table<int, int, custom_hash> mp;

int main(int argc, char* argv[]) {
  ios_base::sync_with_stdio(false);//DON'T
      CC++
  cin.tie(NULL);//DON'T use for interactive
  int seed = atoi(argv[1]);
  //cout << dist(mt) << '\n';
}
```

10.4 XOR Gaussian Elimination [24 lines]

```
const int N=1505;
vector<int> g[N];
signed main()
{
    fast
    int n=1500;
    bitset<N> bit[n+1];
    for(int i=1;i<=n;i++){
        cin >> bit[i];
    }
    vector<int> pivot(n+1,-1);
    int basis[N];
    for(int i=1501;i>=0;i--)
    {
        int in=-1;
        for(int j=1;j<=n;j++)
        {
            if(pivot[j]==-1) continue;
            if(in!=-1 && bit[j][i])
                bit[j]^=bit[in],
                g[j].push_back(in);
            else if(bit[j][i])
                in=j,pivot[j]=i, basis[i]=j;
        }
    }
    return 0;
}
```

11 Number Theory

11.1 Linear Sieve [16 lines]

```
vector<int> pri;
vector<int> lp;   // lowest prime factor
void sieve(int n) {
    lp.assign(n + 1, 0);
    pri.clear();
    for (int i = 2; i <= n; i++) {
        if (lp[i] == 0) {
            lp[i] = i;
            pri.push_back(i);
        }
        for (int p : pri) {
            if (p > lp[i] || 1LL * p * i > n)
                break;
            lp[p * i] = p;
        }
    }
}
```

11.2 Number Theory all concepts [108 lines]

```
#include <bits/stdc++.h>
using namespace std;

using ll = long long;

struct numth
{

    // Struct to hold results of the Extended
        Euclidean Algorithm
    struct exgcd
    {
        ll gcd, x, y; // gcd of a and b,
            coefficients x and y for the
            equation ax + by = gcd(a, b)
    };

    // Extended Euclidean Algorithm to solve ax
        + by = gcd(a, b)
    exgcd exEuclid(ll a, ll b)
    {
        if (b == 0)
        {
            exgcd nd = {a, 1, 0};
            return nd;
        }
        exgcd sml = exEuclid(b, a % b);
        exgcd bg = {sml.gcd, sml.y, sml.x - (a
            / b) * sml.y};
        return bg;
    }
```

```
ll gcd(ll a, ll b)
{
    while (b != 0)
    {
        ll temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
ll EuclidInverse(ll a, ll m)
{
    exgcd sml = exEuclid(a, m);
    if (sml.gcd != 1)
    {
        return -1;
    }
    return (sml.x % m + m) % m;
}

vector<ll> svp;//all primes
vector<ll> svl;//lowest prime
bitset<200005> check;
vector<ll> segmented_sieve(ll l, ll r)
{
    vector<ll> segpr;
    vector<bool> pr(r - l + 5, 1);
    if (l == 1)
    {
        pr[0] = false;
    }
    for (ll i = 0; svp[i] * svp[i] <= r;
        i++)
    {
        ll cur = svp[i];
        ll base = cur * cur;
        if (base < l)
        {
            base = ((l + cur - 1) / cur)
                * cur;
        }
        for (ll j = base; j <= r; j +=
            cur)
        {
            pr[j - l] = false;
        }
    }
    for (ll i = 0; i <= r - l; i++)
    {
        if (pr[i] == 1)
        {
            segpr.push_back(l + i);
        }
    }
    return segpr;
}
ll sumOfDivisors(ll n)
{
    vector<pair<ll, ll>> factors =
        primeFactorization(n);
    ll sum = 1;
    for (auto &factor : factors)
    {
        ll p = factor.first;
        ll a = factor.second;
        ll term = 1;
        for (ll i = 0; i <= a; i++)
        {
            term *= p;
        }
        sum *= (term - 1) / (p - 1);
    }
    return sum;
}
ll CRT(vector<ll> nums, vector<ll> rems)
{
    ll prod = accumulate(nums.begin(),
        nums.end(), 1LL,
        multiplies<ll>());
    ll result = 0;
    for (size_t i = 0; i < nums.size();
        ++i)
    {
        ll pp = prod / nums[i];
        result += rems[i] *
            EuclidInverse(pp, nums[i]) *
            pp;
    }
    return result % prod;
}
};
```

11.3 Phi And Mobius [37 lines]

```
//all of (1-10^6) -> O(nlogn)
int phi[N];
void totient() {
    for (int i = 0; i < N; i++) phi[i] = i;
    for (int i = 2; i < N; i++) {
        if (phi[i] != i) continue;
        for (int j = i; j < N; j += i)
            phi[j] -= phi[j] / i;
    }
}
//10^16 range->O(sqrt(n))
int phiValue(int n)
{
    int ans=1;
    int q=sqrt(n);
    for(int i=2;i<=q;i++)
    {
        if(n%i==0)
        {
            int tem=1;
            while(n%i==0) tem*=i, n/=i;
            ans=ans*tem/i*(i-1);
            q=sqrt(n);
        }
    }
    if(n>1)ans=ans*(n-1);
    return ans;
}
//mobius O(nlogn)
int mob[N];
void mobius()
{
    for(int i=0;i<N;i++)mob[i]=0;
    mob[1]=1;
    for(int i=1;i<N;i++)
        for(int j=i+i;j<N;j+=i)mob[j]-=mob[i];
}
```

11.4 Pollard Rho [88 lines]

```
namespace PollardRho {
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
const int P = 1e6 + 9;
ll seq[P];
int primes[P], spf[P];
inline ll add_mod(ll x, ll y, ll m) {
  return (x += y) < m ? x : x - m;
}
inline ll mul_mod(ll x, ll y, ll m) {
  ll res = __int128(x) * y % m;
  return res;
  // ll res = x * y - (ll)((long double)x * y
      / m + 0.5) * m;
  // return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m) {
  ll res = 1 % m;
  for (; n; n >>= 1) {
    if (n & 1) res = mul_mod(res, x, m);
    x = mul_mod(x, x, m);
  }
  return res;
}
// O(it * (logn)^3), it = number of rounds
    performed
inline bool miller_rabin(ll n) {
  if (n <= 2 || (n & 1 ^ 1)) return (n ==
      2);
  if (n < P) return spf[n] == n;
  ll c, d, s = 0, r = n - 1;
  for (; !(r & 1); r >>= 1, s++) {}
  // each iteration is a round
  for (int i = 0; primes[i] < n && primes[i]
      < 32; i++) {
    c = pow_mod(primes[i], r, n);
    for (int j = 0; j < s; j++) {
      d = mul_mod(c, c, n);
      if (d == 1 && c != 1 && c != (n - 1))
          return false;
      c = d;
    }
    if (c != 1) return false;
  }
  return true;
}
//initialize just one time
void init() {
  int cnt = 0;
  for (int i = 2; i < P; i++) {
    if (!spf[i]) primes[cnt++] = spf[i] = i;
    for (int j = 0, k; (k = i * primes[j]) <
        P; j++) {
      spf[k] = primes[j];
      if (spf[i] == spf[k]) break;
    }
  }
}
// returns O(n^(1/4))
ll pollard_rho(ll n) {
  while (1) {
    ll x = rnd() % n, y = x, c = rnd() % n,
        u = 1, v, t = 0;
    ll *px = seq, *py = seq;
    while (1) {
      *py++ = y = add_mod(mul_mod(y, y, n),
          c, n);
      *py++ = y = add_mod(mul_mod(y, y, n),
          c, n);
      if ((x = *px++) == y) break;
      v = u;
      u = mul_mod(u, abs(y - x), n);
      if (!u) return __gcd(v, n);
      if (++t == 32) {
        t = 0;
        if ((u = __gcd(u, n)) > 1 && u < n)
            return u;
      }
    }
    if (t && (u = __gcd(u, n)) > 1 && u < n)
        return u;
  }
}
vector<ll> factorize(ll n) {
  if (n == 1) return vector <ll>();
  if (miller_rabin(n)) return vector<ll>
      {n};
  vector <ll> v, w;
  while (n > 1 && n < P) {
    v.push_back(spf[n]);
    n /= spf[n];
  }
  if (n >= P) {
    ll x = pollard_rho(n);
    v = factorize(x);
    w = factorize(n / x);
    v.insert(v.end(), w.begin(), w.end());
  }
  return v;
}
}
```

12 String

12.1 2D Hashing [48 lines]

```
#include<bits/stdc++.h>
using namespace std;

const int N = 3e5 + 9;

struct Hashing {
  //just for string with characters >= a
  vector<vector<int>> hs;
  vector<int> PWX, PWY;
  int n, m;
  static const int PX = 3731,  PY = 2999, mod
      = 998244353;
  Hashing() {}
  Hashing(vector<string>& s) {
    n = (int)s.size(), m = (int)s[0].size();
    hs.assign(n + 1, vector<int>(m + 1, 0));
    PWX.assign(n + 1, 1);
    PWY.assign(m + 1, 1);
    for (int i = 0; i < n; i++) PWX[i + 1] =
        1LL * PWX[i] * PX % mod;
    for (int i = 0; i < m; i++) PWY[i + 1] =
        1LL * PWY[i] * PY % mod;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        hs[i + 1][j + 1] = s[i][j] - 'a' + 1;
      }
```

```
    }
    for (int i = 0; i <= n; i++) {
      for (int j = 0; j < m; j++) {
        hs[i][j + 1] = (hs[i][j + 1] + 1LL *
            hs[i][j] * PY % mod) % mod;
      }
    }
    for (int i = 0; i < n; i++) {
      for (int j = 0; j <= m; j++) {
        hs[i + 1][j] = (hs[i + 1][j] + 1LL *
            hs[i][j] * PX % mod) % mod;
      }
    }
  }
  int get_hash(int x1, int y1, int x2, int y2)
      { // 1-indexed
    assert(1 <= x1 && x1 <= x2 && x2 <= n);
    assert(1 <= y1 && y1 <= y2 && y2 <= m);
    x1--;
    y1--;
    int dx = x2 - x1, dy = y2 - y1;
    return (1LL * (hs[x2][y2] - 1LL *
        hs[x2][y1] * PWY[dy] % mod + mod) %
        mod -
        1LL * (hs[x1][y2] - 1LL * hs[x1][y1]
            * PWY[dy] % mod + mod) % mod *
            PWX[dx] % mod + mod) % mod;
  }
  int get_hash() {
    return get_hash(1, 1, n, m);
  }
};
```

12.2 Dynamic Aho Corrasic [148 lines]

```
#include <bits/stdc++.h>
using namespace std;

struct AhoCorasick
{
    struct Node
    {
        int children[26], go[26], fail;
        int patternCount;              // Count
            patterns ending here
        vector<int> patternIndices;    //
            Stores pattern IDs

        Node()
        {
            memset(children, -1,
                sizeof(children));
            memset(go, -1, sizeof(go));
            fail = -1;
            patternCount = 0;
        }
    };

    vector<Node> nodes;
    int root;

    AhoCorasick()
    {
        root = 0;
        nodes.push_back(Node());
    }

    int charToIndex(char c)
    {
        return c - 'a';  // assumes lowercase
            'a' to 'z'
    }

    // Insert pattern, assign an ID
    void insert(const string &s, int id)
    {
        int curr = root;
        for (char c : s)
        {
            int i = charToIndex(c);
            if (nodes[curr].children[i] == -1)
            {
                nodes[curr].children[i] =
                    nodes.size();
                nodes.push_back(Node());
            }
            curr = nodes[curr].children[i];
        }
        nodes[curr].patternCount++;
        nodes[curr].patternIndices.
            push_back(id);
    }

    void build()
    {
        queue<int> q;
        nodes[root].fail = root;

        // Initialize go transitions for root
        for (int i = 0; i < 26; ++i)
        {
            if (nodes[root].children[i] != -1)
            {
                int child =
                    nodes[root].children[i];
                nodes[child].fail = root;
                q.push(child);
                nodes[root].go[i] = child;
            }
            else
            {
                nodes[root].go[i] = root;
            }
        }

        while (!q.empty())
        {
            int curr = q.front();
            q.pop();

            for (int i = 0; i < 26; ++i)
            {
                int child =
                    nodes[curr].children[i];
                if (child != -1)
                {
                    int fail =
                        nodes[curr].fail;
                    while (fail != root &&
                        nodes[fail].
                        children[i] == -1)
                        fail =
                            nodes[fail].fail;
                    if (nodes[fail].
                        children[i] != -1)
                        fail = nodes[fail].
                            children[i];
                    else
                        fail = root;

                    nodes[child].fail = fail;
                    nodes[child].patternCount
                        += nodes[fail].
                        patternCount;

                    // Merge pattern counts and
                        IDs
                    for (int id :
                        nodes[fail].
                        patternIndices)
                        nodes[child].
                            patternIndices.
                            push_back(id);

                    q.push(child);

                    nodes[curr].go[i] = child;
                }
                else
                {
                    nodes[curr].go[i] =
                        nodes[nodes[curr].
                        fail].go[i];
                }
            }
        }
    }

    // Count total occurrences
    int searchCount(const string &text)
    {
        int curr = root;
        int count = 0;

        for (char c : text)
        {
            int i = charToIndex(c);
            curr = nodes[curr].go[i];
            count += nodes[curr].patternCount;
        }
        return count;
    }

    // Get matched pattern ids with position
    vector<pair<int, int>>
        searchWithIndex(const string &text)
    {
        vector<pair<int, int>> matches;
        int curr = root;

        for (int j = 0; j < text.size(); ++j)
        {
            int i = charToIndex(text[j]);
            curr = nodes[curr].go[i];

            int temp = curr;
            while (temp != root)
            {
                for (int id : nodes[temp].
                    patternIndices)
                    matches.push_back({id,
                        j});
                temp = nodes[temp].fail;
            }
        }
        return matches;
    }
};
```

12.3 KMP [54 lines]

```
vector<int> prefix_kmp(string &s)
{
    int n = s.size();
    vector<int> pi(n, 0);
    for (int i = 1; i < n; i++)
    {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j
            - 1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
vector<vector<int>> automation(string &s)
{
    vector<int> pre = prefix_kmp(s);
    int m = s.size();
    vector<vector<int>> nxt(m,
        vector<int>(26, 0));
    for (int st = 0; st < m; st++)
    {
        for (int ci = 0; ci < 26; ci++)
        {
            char ch = char('A' + ci);
            int j = st;
            while (j > 0 && s[j] != ch) j =
                pre[j - 1];
            if (s[j] == ch) j++;
            nxt[st][ci] = j;
        }
    }
    return nxt;
}

void kmp_base(string &demo, string &pattern)
{
    vector<int> prefix_arr =
        prefix_kmp(pattern);
    int i = 0, j = 0;
    vector<int> positions;
    while (i < demo.size())
    {
        if (demo[i] == pattern[j]) i++,j++;
        if (j == pattern.size())
        {
            int pos = i - j;
            positions.push_back(pos); // Store
                the start index of the match
```

```
            j = prefix_arr[j - 1];     //
                Continue searching for next
                occurrences
        }
        else if (i < demo.size() && demo[i]
            != pattern[j])
        {
            if (j != 0)j = prefix_arr[j - 1];
            else i++;
        }
    }
}
```

12.4 Manachar [30 lines]

```
int p[2*N];
void manacher (string s)
{
    int n=s.size();
    char ss[2*n+2];
    ss[2*n+1]='\0';
    for(int i=0; i<2*n+1; i++)
    {
        p[i]=0;
        if(i%2==0)ss[i]='#';
        else ss[i]=s[i/2];
    }
    int l = 0, r = 0;
    for(int i = 0; i <=2*n; i++)
    {
        p[i] = max(0, min(r - i, p[l + (r -
            i)]));
        while((p[i]+i<=2*n)&&(i-p[i]>=
            0)&&(ss[i - p[i]] == ss[i +
            p[i]]))
        {
            p[i]++;
        }
        if(i + p[i] > r)
        {
            l = i - p[i], r = i + p[i];
        }
    }
}
bool check(int l,int r)
{
    return p[l+r+1]>r-l;
}
```

12.5 Palindromic Tree [40 lines]

```
//just create one PTree for all the test cases
    and reset everytime
struct PTree{
    //// N should be set so that N >=
        max_string_length + 5
  int S[N],nx[N];
  int head[N],nxt[N],ch[N];
    int link[N],len[N],cnt[N],lst,nd,n,e;
    ll total;
    PTree(){reset();}
  inline int newnode(int L){
    cnt[nd]=0,len[nd]=L,head[nd]=0;
    return nd++;
  }
  //O(1)
  inline void reset(){
    total=e=nd=n=0,lst=1;
    newnode(0),newnode(-1);
    S[0]=-1,link[0]=1;
  }
  inline int getLink(int v){
    while(S[n-len[v]-1]!=S[n]) v=link[v];
    return v;
  }
  inline void add(int c){
    S[++n]=c;
    int cur=getLink(lst),i,j;
    for(i=head[cur];i;i=nxt[i]) if(nx[i]==c)
        break;
    if(!i){
      int now=newnode(len[cur]+2);
      int x=getLink(link[cur]);
      for(j=head[x];j;j=nxt[j]) if(nx[j]==c)
          break;
      if(j) link[now]=ch[j];else link[now]=0;
      nxt[++e]=head[cur];
      head[cur]=e;
      nx[e]=c;ch[e]=now;
      cnt[now]=cnt[link[now]]+1;
      lst=now;
    }else lst=ch[i];
        total+=cnt[lst];
  }
};
```

12.6 String Hashing [49 lines]

```
struct SimpleHash {
    long long len, base, mod;
    vector<int> P, H, R;
    SimpleHash() {}
    SimpleHash(string str, long long b, long
        long m) {
        base = b, mod = m, len = str.size();
        P.resize(len + 4, 1), H.resize(len +
            3, 0), R.resize(len + 3, 0);
        for (long long i = 1; i <= len + 3;
            i++)
            P[i] = ((ll)P[i - 1] * base) %
                mod;
        for (long long i = 1; i <= len; i++)
            H[i] = ((ll)H[i - 1] * base +
                str[i - 1]+1007) % mod;
        for (long long i = len; i >= 1; i--)
            R[i] = ((ll)R[i + 1] * base +
                str[i - 1]+1007) % mod;
    }
    inline long long range_hash(long long l,
        long long r) {
        long long hashval = ((ll)H[r + 1] -
            ((ll)P[r - l + 1] * (ll)H[l] %
            mod))%mod;
        return (hashval < 0 ? hashval + mod :
            hashval);
    }
    inline long long reverse_hash(long long
        l, long long r) {
        long long hashval = (ll)R[l + 1] -
            ((ll)P[r - l + 1] * (ll)R[r + 2] %
            mod);
        return (hashval < 0 ? hashval + mod :
            hashval);
    }
};
struct DoubleHash {
    SimpleHash sh1, sh2;
    DoubleHash() {}
    DoubleHash(string str) {
        sh1 = SimpleHash(str, 1949313259,
            2091573227);
        sh2 = SimpleHash(str, 1997293877,
            2117566807);
    }
    long long concate(DoubleHash& B, long long
        l1 , long long r1 , long long l2 ,
        long long r2) {
        long long len1 = r1 - l1+1 , len2 = r2
            - l2+1;
        long long x1 = sh1.range_hash(l1, r1)
            ,
        x2 = B.sh1.range_hash(l2, r2);
        x1 = (x1 * B.sh1.P[len2]) %
            2091573227;
        long long newx1 = (x1 + x2) %
            2091573227;
        x1 = sh2.range_hash(l1, r1);
        x2 = B.sh2.range_hash(l2, r2);
        x1 = (x1 * B.sh2.P[len2]) %
            2117566807;
        long long newx2 = (x1 + x2) %
            2117566807;
        return (newx1 << 32) ^ newx2;
    }
    inline long long range_hash(long long l,
        long long r) {
        return (sh1.range_hash(l, r) << 32) ^
            sh2.range_hash(l, r);
    }
    inline long long reverse_hash(long long
        l, long long r) {
        return (sh1.reverse_hash(l, r) << 32)
            ^ sh2.reverse_hash(l, r);
    }
};
```

12.7 Suffix Array [105 lines]

```
vector<pair<int,int>>suflcp(vector<int>ss)
    //string s -- vector<int>s (both works)
{
    int n=ss.size();
    //ranking elements
    auto s=ss;
    vector<pair<int,int>>rnk;
    //map<int,vector<int>>rnk;
    for(int i=0; i<n; i++)
        rnk.push_back({s[i],i});
    sort(all(rnk));
    int pre=-1,prev=-M;
    for(auto it:rnk)
    {
        if(it.first!=prev)pre++,prev=it.first;
        s[it.second]=pre;
    }//ranking ends
    int q=1,co=1;
    while(q<n)co++,q*=2;
    int tra[n][co];
    vector<pair<pair<int,int>,int>>str,tem[n+
        3];
    for(int i=0; i<n; i++)
        str.push_back({{s[i],-1},i});
    //sort(all(str));
    for(int i=0; i<n; i++) tem[str[i].first.
        second+1].push_back(str[i]);
    str.clear();
    for(int i=0; i<n+3; i++)
    {
        for(auto it:tem[i])str.push_back(it);
        tem[i].clear();
    }
    for(int i=0; i<n; i++) tem[str[i].first.
        first+1].push_back(str[i]);
    str.clear();
    for(int i=0; i<n+3; i++)
    {
        for(auto it:tem[i])str.push_back(it);
        tem[i].clear();
    }
    //sort end
    int k=-1;
    pair<int,int>p= {-inf,-inf};
    for(auto it:str)
    {
        if(it.first!=p)k++,p=it.first;
        tra[it.second][0]=k;
    }
    q=1;
    for(int i=1; i<co; i++)
    {
        str.clear();
        for(int j=0; j<n; j++)
        {
            if(j+q>=n)
                str.push_back({{tra[j][i-1],-
                    1},j});
            else str.push_back({{tra[j][i-1],
                tra[j+q][i-1]},j});
        }
        //sort(all(str));
        for(int i=0; i<n; i++)
            tem[str[i].first.second+1].
            push_back(str[i]);
        str.clear();
        for(int i=0; i<n+3; i++)
        {
            for(auto
                it:tem[i])str.push_back(it);
            tem[i].clear();
        }
        for(int i=0; i<n; i++) tem[str[i].
            first.first+1].push_back(str[i]);
        str.clear();
        for(int i=0; i<n+3; i++)
        {
            for(auto
                it:tem[i])str.push_back(it);
            tem[i].clear();
        }
        //sort end
        int k=-1;
        pair<int,int>p= {-inf,-inf};
        for(auto it:str)
        {
            if(it.first!=p)k++,p=it.first;
```

```
            tra[it.second][i]=k;
        }
        q*=2;
    }
    vector<pair<int,int>>ans(n);
    for(int i=0; i<n; i++)
        ans[tra[i][co-1]].first=i;
    //lcp calculation
    vector<int> rank(n, 0);
    for (int i = 0; i < n;
        i++)rank[ans[i].first] = i;
    k = 0;
    //vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++)
    {
        if (rank[i] == n - 1)
        {
            k = 0;
            continue;
        }
        int j = ans[rank[i] + 1].first;
        while (i + k < n && j + k < n &&
            s[i+k] == s[j+k]) k++;
        ans[rank[i]].second = k;
        if (k) k--;
    }
    return ans; //ans[n-1].second=0;
}
vector<pair<int,int>>suflcp(string ss)
{
    int n=ss.size();
    vector<int>sss;
    for(int i=0;i<n;i++)sss.push_back(ss[i]);
    return suflcp(sss);
}
```

12.8 Z Algorithm [14 lines]

```
vector<int> Zfunction(string &s)
{
    int n = s.size();
    vector<int> z(n, 0);
    int l = 0, r = 0;
    for (int i = 1; i < n; ++i)
    {
        if (i <= r) z[i] = min(r - i + 1, z[i
            - l]);
        while (i + z[i] < n && s[z[i]] == s[i
            + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i +
            z[i] - 1;
    }
    z[0] = n;
    return z;
}
```

13 Random

13.1 Combinatorics

- $\sum_{k=0}^{n} \binom{n-k}{k} = Fib_{n+1}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k\binom{n}{k} = n\binom{n-1}{k-1}$
- Number of binary sequences of length n such that no two 0's are adjacent $= Fib_{n+1}$
- Number of non-negative solution of $x_1 + x_2 + x_3 + ... + x_k = n$ is $\binom{n+k-1}{n}$

13.1.1 Catalan Number

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$
- $C_0 = 1, C_1 = 1, C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786$
- Number of correct bracket sequences consisting of n opening brackets.
- Number of ways to completely parenthesize n+1 factors.
- The number of triangulations of a convex polygon with +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 monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal
- Number of permutation of length n that can be stack sorted.
- The number of non-crossing partitions of a set of n elements.
- The number of rooted full binary tree with n+1 leaves.
- The number of Dyck words of length 2n. A string consisting of n X's and n Y's such that no string prefix has more Y's than X's.
- Number of permutation of length n with no three-term increasing subsequence.
- Number of ways to tile a stairstep shape of height n with n rectangle.
- $C_n^k = \frac{k+1}{n+1}\binom{2n-k}{n-k}$ denote the number of bracket sequences of size 2n with the first k elements being (.
- $N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$
- The number of expressions containing n pairs of correct parentheses, which contain k distinct nestings. $N(4,2) = 6$ $()((()))$, $(())(())$, $(()(()))$, $(((())))$, $((()))()$, $(((())))$
- The number of paths from (0,0) to (2n, 0) with steps only northeast and southeast, not staying below the x-axis with k peaks. And sum of all number of peaks is Catalan number.

13.1.2 Stirling Number of the First Kind

- Count permutation according to their number of cycles.
- $S(n,k)$ count the number of permutation of n elements with k disjoint cycles.
- $S(n,k) = (n-1) \times S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,1) = (n-1)!$
- $S(n,n-1) = \binom{n}{2}$
- $\sum_{k=0}^{n} S(n,k) = n!$

13.1.3 Stirling Numbers of the Second Kind

- Number of ways to partition a set of n objects into k non-empty subsets.
- $S(n,k) = k * S(n-1,k) + S(n-1,k-1), S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,2) = 2^{n-1} - 1$
- $S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- $S(n,k) * k!$ = number of ways to color n nodes using colors from 1 to k such that each color is used at least once.

13.1.4 Bell Number

- Counts the number of partitions of a set.
- $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k} * B_k$
- $B_n = \sum_{k=0}^{n} S(n,k)$, where S is Stirling number of second kind.
- The number of multiplicative partitions of a square free number with i prime factors is the i-th Bell number.
- $B(p^m+n) \equiv mB(n)+B(n+1) \pmod p$
- If a deck is shuffled by removing and reinserting the top card n times, there are n^n possible shuffles. The number of shuffles that return the deck to its original order is B_n, so the probability of returning to the original order is B_n/n^n.

13.1.5 Lucas Theorem

- If p is prime then $\binom{p^a}{k} \equiv 0 \pmod p$
- For non-negative integers m and n and a prime p:
  $\binom{m}{n} = \prod_{i=0}^{k}\binom{m_i}{n_i} \pmod p$ where
  $m = m_k p^k + m_{k-1}p^{k-1} + ... + m_1 p + m_0$
  $n = n_k p^k + n_{k-1}p^{k-1} + ... + n_1 p + n_0$ are the base p expansion.

13.1.6 Derangement

- A permutation such that no element appears in its original position.
- $d(n) = (n-1) * (d(n-1) + d(n-2)), d(0) = 1, d(1) = 0$
- $d(n) = nd(n-1) + (-1)^n = \lfloor \frac{n!}{e} \rfloor, n \geq 1$

13.1.7 Burnside Lemma

Given a group G of symmetries and a set X, the number of elements of X up to symmetry equals

$$\frac{1}{|G|}\sum_{g \in G}|X^g|$$

where X^g are the elements fixed by $g(g.x = x)$ If f(n) counts "configurations" of some sort of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k)$$

13.1.8 Eulerian Number

- $E(n,k)$ is the number of permutations of the numbers 1 to n in which exactly k elements are greater than the previous element.
- $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k), E(n,0) = E(n,n-1) = 1$
- $E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$
- $E(n,k) = E(n,n-1-k)$
- $E(0,k) = [k=0]$
- $E(n,1) = 2^n - n - 1$

13.2 Number Theory

13.2.1 Mobius Function and Inversion

- define $\mu(n)$ as the sum of the primitive nth roots of unity depending on the factorization of n into prime factors:

$$\mu(x) = \begin{cases} 0 & \text{n is not square free} \\ 1 & \text{n has even number of prime factors} \\ -1 & \text{n has odd number of prime factors} \end{cases}$$

- Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

- $\sum_{d|n} \mu(d) = [n=1]$
- $\phi(n) = \sum_{d|n} \mu(d).\frac{n}{d} = n \sum_{d|n} \frac{\mu(d)}{d} = \sum_{d|n} d.\mu(\frac{n}{d})$
- $a|b \to \phi(a)|\phi(b)$
- $\phi(mn) = \phi(m).\phi(n).\frac{d}{\phi(d)}$ where $d = gcd(m,n)$
- $\phi(n^m) = n^{m-1}\phi(n)$
- $\sum_{i=1}^{n} [gcd(i,n) = k] = \phi(\frac{n}{k})$
- $\sum_{i=1}^{n} gcd(i,n) = \sum_{d|n} d.\phi(\frac{n}{d})$
- $\sum_{i=1}^{n} \frac{1}{gcd(i,n)} = \sum_{d|n} \frac{1}{d}.\phi(\frac{n}{d}) = \frac{1}{n} \sum_{d|n} d.\phi(d)$
- $\sum_{i=1}^{n} \frac{i}{gcd(i,n)} = \frac{n}{2}.\sum_{d|n} \frac{1}{d}.\phi(\frac{n}{d}) = \frac{n}{2}.\frac{1}{n} \sum_{d|n} d.\phi(d)$
- $\sum_{i=1}^{n} \frac{n}{gcd(i,n)} = 2.\sum_{i=1}^{n} \frac{i}{gcd(i,n)} - 1$

13.2.2 GCD and LCM

- gcd(a,b) = gcd(b, a mod b)
- If $a|b.c$, and gcd(a,b) = d, then $(a/d)|c$.
- GCD is a multiplicative function.
- gcd(a, lcm(b,c)) = lcm(gcd(a,b), gcd(a,c))
- $gcd(n^a - 1, n^b - 1) = n^{gcd(a,b)} - 1$

13.2.3 Gauss Circle Theorem

- Determine the number of lattice points in a circle centered at the origin with radius r.
- number of pairs (m,n) such that $m^2 + n^2 \leq r^2$
- $N(r) = 1 + 4\sum_{i=0}^{\infty} (\lfloor \frac{r^2}{4i+1} \rfloor - \lfloor \frac{r^2}{4i+3} \rfloor)$

13.2.4 Pick's Theorem

According to Pick's Theorem We can calculate the area of any polygon by just counting the number of Interior and Boundary lattice points of that polygon. If number of interior points are I and number of boundary lattice points are B then Area (A) of polygon will be:

$$Area = I + B/2 - 1$$

where I is the number of points in the interior shape, B stands for the number of points on the boundary of the shape.

13.2.5 Formula Cheatsheet

- $\sum_{i=1}^{n} = \frac{1}{m+1}[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}((i+1)^{m+1} - i^{m+1} - (m+1)i^m)]$
- $\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1$
- $\sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$
- $H_n = \sum_{i=1}^{n} \frac{1}{n}, \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$
- $\sum_{k=0}^{n} \binom{r+k}{k} = \binom{r+n+1}{n}$