Team Notebook

June 21, 2022

Contents

1 Brute Force

1.1 Power Set [SA]

```
void power_set(vector<int>& v, int n, int i = 0) {
    if (i == n) {
        // print (v)
        return;
    }
    v.push_back(i);
    power_set(v, n, i + 1);
    v.pop_back();
    power_set(v, n, i + 1);
}
```

2 Combinatorics

2.1 nCr mod P in O(1) [SK]

```
const int N = 1e5 + 5;

// array to store inverse of 1 to N
ll factorialNumInverse[N + 1];

// array to precompute inverse of 1! to N!
ll naturalNumInverse[N + 1];

// array to store factorial of first N numbers
ll fact[N + 1];

// Function to precompute inverse of numbers
void InverseofNumber(ll p)
{
    naturalNumInverse[0] = naturalNumInverse[1] = 1;
    for (int i = 2; i <= N; i++)
        naturalNumInverse[i] = naturalNumInverse[p % i] * (p
             - p / i) % p;
}
// Function to precompute inverse of factorials
void InverseofFactorial(ll p)
{
    factorialNumInverse[0] = factorialNumInverse[1] = 1;

    // precompute inverse of natural numbers
    for (int i = 2; i <= N; i++)
        factorialNumInverse[i] = (naturalNumInverse[i] *
             factorialNumInverse[i - 1]) % p;
}

// Function to calculate factorial of 1 to N
void factorial(ll p)
{
    fact[0] = 1;

    // precompute factorials
    for (int i = 1; i <= N; i++)
    {
        fact[i] = (fact[i - 1] * i) % p;
    }
}

// Function to return nCr % p in O(1) time
ll Binomial(ll N, ll R, ll p)
{
    // n C r = n!*inverse(r!)*inverse((n-r)!)
    ll ans = ((fact[N] * factorialNumInverse[R])
              % p * factorialNumInverse[N - R])
             % p;
    return ans;
}
```

2.2 nCr [SA]

```
const int N = 105, R = 85;
int64_t NCR[N][R];

void genNCR() {
    for (int i = 0; i < N; ++i) {
        NCR[i][0] = 1;
        if (i < R) NCR[i][i] = 1;
    }
    for (int i = 1; i < N; ++i) {
        for (int j = 1; j <= i; ++j) {
            NCR[i][j] = NCR[i - 1][j - 1] + NCR[i - 1][j];
        }
    }
}
```

3 Data Structures

3.1 2D Prefix Sum [SA]

```
const int N = 1000, M = 500;
int a[N + 1][M + 1], pref[N + 1][M + 1];

// 1-based
void build() {
    for (int i = 1; i <= N; ++i) {
        for (int j = 1; j <= M; ++j) {
            pref[i][j] = pref[i - 1][j] + pref[i][j - 1] -
                 pref[i - 1][j - 1] + a[i][j];
        }
    }
}
// top_left(i, j), right_bottom(k, l)
auto query(int i, int j, int k, int l) {
    return pref[k][l] - pref[i - 1][l] - pref[k][j - 1] +
         pref[i - 1][j - 1];
}
```

3.2 Disjoint Set Union [SA]

```
const int N = 100001;
int parent[N], comp_size[N];
int components = 0;

void make_set(int u) {
    parent[u] = u;
    comp_size[u] = 1;
    ++components;
}

int get_size(int u) {
    return comp_size[find(u)];
}

int find(int u) {
    if (u == parent[u]) return u;
    return parent[u] = find(parent[u]);
}

void unite(int u, int v) {
    u = find(u), v = find(v);
    if (u != v) {
        if (comp_size[u] < comp_size[v]) {
            swap(u, v);
        }
        parent[v] = u;
        comp_size[u] += comp_size[v];
        --components;
    }
}
```

3.3 DSU [RH]

```
int dsuparent[100000];
int dsurank[100000];
void makedsu()
{
    for (int i = 0; i < 100000; i++)
 {
        dsuparent[i] = i;
        dsurank[i] = 0;
    }
}
int findparent(int node)
{
    if (node == dsuparent[node])
 {
        return node;
    }
    return dsuparent[node] = findparent(dsuparent[node]);
}
void dsuunion(int u, int v)
{
    u = findparent(u);
    v = findparent(v);
    if (dsurank[u] < dsurank[v])
 {
        dsuparent[u] = v;
    }
    else if (dsurank[u] < dsurank[v])
 {
        dsuparent[v] = u;
    }
    else
 {
        dsuparent[v] = u;
        dsurank[u]++;
    }
}
```

3.4 Segment Tree Lazy Updates [SK]

```
const int N = 1e5 + 5;
ll v[4 * N];
ll add[4 * N];
int arr[N];

void push(int cur)
{
    add[cur * 2] += add[cur];
    add[cur * 2 + 1] += add[cur];
    add[cur] = 0;
}
void build(int cur, int l, int r)
{
    if (l == r)
    {
        v[cur] = arr[l];
        return;
    }

    int mid = l + (r - l) / 2;

    build(cur * 2, l, mid);
    build(cur * 2 + 1, mid + 1, r);

    v[cur] = v[cur * 2] + v[cur * 2 + 1];

    return;
}
ll query(int cur, int l, int r, int x, int y)
{

    if (x > r || y < l)
    {
        return 0;
    }
    if (l == r)
    {
        return v[cur] + add[cur];
    }
    if (l == x && r == y)
    {
        return v[cur] + add[cur] * (r - l + 1);
    }
    int mid = l + (r - l) / 2;

    v[cur] += add[cur] * (r - l + 1);
    push(cur);

    ll left = query(cur * 2, l, mid, x, min(mid, y));
    ll right = query(cur * 2 + 1, mid + 1, r, max(mid + 1, x)
        , y);

    ll res = 0;
    res = left + right;
    return res;
}
void update(int cur, int l, int r, int s, int e, int val)
{
    if (l == s && r == e)
    {
        add[cur] += val;
        return;
    }

    if (s > r || e < l)
    {
        return;
    }

    int mid = l + (r - l) / 2;

    push(cur);

    update(cur * 2, l, mid, s, min(e, mid), val);
    update(cur * 2 + 1, mid + 1, r, max(s, mid + 1), e, val);

    v[cur] = (v[cur * 2] + add[cur * 2] * (mid - l + 1)) + (v
        [cur * 2 + 1] + add[cur * 2 + 1] * (r - mid));
    return;
}
```

3.5 Segment Tree Range Additions [SA]

```
const int N = 100001;
int a[N], tree[4 * N], lazy[4 * N];

// 0-based indexing
void build(int node, int tL, int tR) {
    if (tL == tR) {
        tree[node] = a[tL];
        return;
    }
    int mid = (tL + tR) / 2;

    build(2 * node, tL, mid);
    build(2 * node + 1, mid + 1, tR);

    tree[node] = tree[2 * node] + tree[2 * node + 1];
}

void update(int node, int tL, int tR, int qL, int qR, int
    val) {
```

```
    if (lazy[node] != 0) {
        tree[node] += (tR - tL + 1) * lazy[node];

        if (tL != tR) {
            lazy[2 * node] += lazy[node];
            lazy[2 * node + 1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if (tR < qL || tL > qR) {
        return;
    }
    if (tL >= qL && tR <= qR) {
        tree[node] += (tR - tL + 1) * val;
        if (tL != tR) {
            lazy[2 * node] += val;
            lazy[2 * node + 1] += val;
        }
        return;
    }

    int mid = (tL + tR) / 2;

    update(2 * node, tL, mid, qL, qR, val);
    update(2 * node + 1, mid + 1, tR, qL, qR, val);

    tree[node] = (tree[2 * node] + tree[2 * node + 1]);
}

int query(int node, int tL, int tR, int l, int r) {
    if (lazy[node] != 0) {
        tree[node] += (tR - tL + 1) * lazy[node];
        if (tL != tR) {
            lazy[2 * node] += lazy[node];
            lazy[2 * node + 1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if (tR < l || tL > r) {
        return 0;
    }

    if (l <= tL && tR <= r) {
        return tree[node];
    }
    int mid = (tL + tR) / 2;

    int QL = query(2 * node, tL, mid, l, r);
    int QR = query(2 * node + 1, mid + 1, tR, l, r);

    return QL + QR;
}
```

3.6 Segment Tree Range Assignments [SA]

```
const int N = 100001;
int a[N], tree[4 * N], lazy[4 * N];

// 0-based indexing
void build(int node, int tL, int tR) {
    if (tL == tR) {
        tree[node] = a[tL];
        return;
    }
    int mid = (tL + tR) / 2;

    build(2 * node, tL, mid);
    build(2 * node + 1, mid + 1, tR);

    tree[node] = tree[2 * node] + tree[2 * node + 1];
}

void update(int node, int tL, int tR, int qL, int qR, int
     val) {
    if (lazy[node] != 0) {
        tree[node] += (tR - tL + 1) * lazy[node];

        if (tL != tR) {
            lazy[2 * node] += lazy[node];
            lazy[2 * node + 1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if (tR < qL || tL > qR) {
        return;
    }
    if (tL >= qL && tR <= qR) {
        tree[node] += (tR - tL + 1) * val;
        if (tL != tR) {
            lazy[2 * node] += val;
            lazy[2 * node + 1] += val;
        }
        return;
    }

    int mid = (tL + tR) / 2;

    update(2 * node, tL, mid, qL, qR, val);
    update(2 * node + 1, mid + 1, tR, qL, qR, val);

    tree[node] = (tree[2 * node] + tree[2 * node + 1]);
}

int query(int node, int tL, int tR, int l, int r) {
    if (lazy[node] != 0) {
        tree[node] += (tR - tL + 1) * lazy[node];
        if (tL != tR) {
            lazy[2 * node] += lazy[node];
            lazy[2 * node + 1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if (tR < l || tL > r) {
        return 0;
    }

    if (l <= tL && tR <= r) {
        return tree[node];
    }
    int mid = (tL + tR) / 2;

    int QL = query(2 * node, tL, mid, l, r);
    int QR = query(2 * node + 1, mid + 1, tR, l, r);

    return QL + QR;
}
```

3.7 Segment Tree [RH]

```
const long long int maxn = 100010;
long long int segtree[4 * maxn];
void build_seg_tree(long long int a[], long long int v, long
     long int tl, long long int tr)
{
    if (tl == tr)
    {
        segtree[v] = a[tl];
    }
    else
    {
        long long int tm = (tl + tr) / 2;
        build_seg_tree(a, v * 2, tl, tm);
        build_seg_tree(a, v * 2 + 1, tm + 1, tr);
        segtree[v] = segtree[v * 2] + segtree[v * 2 + 1];
    }
}
long long int get(long long int v, long long int tl, long
     long int tr, long long int l, long long int r)
```

```
{
    if (l > r)
    {
        return 0;
    }
    if (l == tl && r == tr)
    {
        return segtree[v];
    }
    long long int tm = (tl + tr) / 2;
    return get(v * 2, tl, tm, l, min(r, tm))
        + get(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r);
}
void update_seg_tree(long long int v, long long int tl, long
      long int tr, long long int pos, long long int new_val)
    {
    if (tl == tr)
    {
        segtree[v] = new_val;
    }
    else
    {
        long long int tm = (tl + tr) / 2;
        if (pos <= tm)
        {
            update_seg_tree(v * 2, tl, tm, pos, new_val);
        }
        else
        {
            update_seg_tree(v * 2 + 1, tm + 1, tr, pos,
                 new_val);
        }
        segtree[v] = segtree[v * 2] + segtree[v * 2 + 1];
    }
}
```

3.8 Segment Tree [SA]

```
const int MX_N = 5 + 1e5;
int a[MX_N], tree[4 * MX_N];


void build(int node, int tL, int tR) {
    if (tL == tR) {
        tree[node] = a[tL];
        return;
    }
    int mid = (tL + tR) / 2;
    int left = 2 * node, right = 2 * node + 1;
    build(left, tL, mid);
    build(right, mid + 1, tR);
    tree[node] = min(tree[left], tree[right]);
}

void update(int node, int tL, int tR, int i, int v) {
    if (tL >= i && tR <= i) {
        tree[node] = v;
        return;
    }
    if (tR < i || tL > i) return;

    int mid = (tL + tR) / 2;
    int left = 2 * node, right = 2 * node + 1;
    update(left, tL, mid, i, v);
    update(right, mid + 1, tR, i, v);
    tree[node] = min(tree[left], tree[right]);
}

int query(int node, int tL, int tR, int qL, int qR) {
    if (tL >= qL && tR <= qR) {
        return tree[node];
    }
    if (tR < qL || tL > qR) {
        return INT_MAX;
    }
    int mid = (tL + tR) / 2;
    int QL = query(2 * node, tL, mid, qL, qR);
    int QR = query(2 * node + 1, mid + 1, tR, qL, qR);
    return min(QL, QR);
}
```

3.9 SegTree Lazy Propagation [RH]

```
const long long int maxn = 100010;
long long int segtree[4 * maxn];
long long int lazy[4 * maxn];
long long int a[maxn];
void fix(int n)
{
    for (int i = 0; i < 4 * n; i++)
    {
        lazy[i] = 0;
        segtree[i] = 0;
    }
}
void build_seg_tree(long long int i, long long int lo, long
     long int hi)
{
    if (lo == hi)
    {
        segtree[i] = a[i];
        lazy[i] = 0;
        return;
    }
    long long int mid = (lo + hi) / 2;
    build_seg_tree(2 * i, lo, mid);
    build_seg_tree(2 * i + 1, mid + 1, hi);
    segtree[i] = segtree[2 * i] + segtree[2 * i + 1];
    lazy[i] = 0;
}
void push(long long int i)
{
    lazy[2 * i] += lazy[i];
    lazy[2 * i + 1] += lazy[i];
    lazy[i] = 0;
}
void update(long long int i, long long int lo, long long int
      hi, long long int l, long long int r, long long int
     val)
{
    if (r<lo || l>hi)
    {
        return;
    }
    if (l <= lo && hi <= r)
    {
        lazy[i] += val;
        return;
    }
    push(i);
    long long int mid = (lo + hi) / 2;
    update(2 * i, lo, mid, l, r, val);
    update(2 * i + 1, mid + 1, hi, l, r, val);
    segtree[i] = segtree[2 * i] + (mid - lo + 1) * lazy[2 * i
         ] + segtree[2 * i + 1] + (hi - mid) * lazy[2 * i +
         1];
}
long long int query(long long int i, long long int lo, long
     long int hi, long long int l, long long int r)
{
    if (r<lo || l>hi)
    {
        return 0;
    }
    if (l <= lo && hi <= r)
    {
        return segtree[i] + lazy[i] * (hi - lo + 1);
    }
```

```
    push(i);
    long long int mid = (lo + hi) / 2;
    long long int ans = query(2 * i, lo, mid, l, r) +
        query(2 * i + 1, mid + 1, hi, l, r);
    segtree[i] = segtree[2 * i] + (mid - lo + 1) * lazy[2 * i
        ] + segtree[2 * i + 1] + (hi - mid) * lazy[2 * i +
        1];
    return ans;
}
```

3.10 Sparse Table [SA]

```
const int N = 100001, LG = 18;
int st[N][LG];

void sparse_table(vector<int>& a, int n) {
    for (int i = 0; i < n; ++i) {
        st[i][0] = a[i];
    }

    for (int j = 1; j < LG; ++j) {
        for (int i = 0; i + (1 << j) - 1 < N; ++i) {
            st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1)
                )][j - 1]);
        }
    }
}

int rmq(int L, int R) {
    int lg = __lg(R - L + 1);
    return min(st[L][lg], st[R - (1 << lg) + 1][lg]);
}
```

4 Dynamic Programming

4.1 Longest Increasing Subsequence [SA]

```
const int MX_N = 1005;
int dp[MX_N];

int LIS(const vector<int>& a, int n, int i) {
    if (dp[i] != -1) return dp[i];

    int ans = 0;
    for (int j = i + 1; j < n; ++j) {
        if (a[j] > a[i]) {
            ans = max(ans, LIS(a, n, j));
        }
    }
    return dp[i] = 1 + ans;
}

int LIS(const vector<int>& a, int n) {
    int ans = 0;
    memset(dp, -1, sizeof dp);
    for (int i = 0; i < n; ++i) {
        ans = max(ans, LIS(a, n, i));
    }
    return ans;
}
```

5 Geometry

5.1 Convex Hull

```
int cross_product(pair<int, int>& O, pair<int, int>& A, pair
    <int, int>& B) {
 return (A.first - O.first) * (B.second - O.second) - (A.
     second - O.second) * (B.first - O.first);
}

vector<pair<int, int>> convex_hull(vector<pair<int, int>> A)
    {
 int n = A.size(), k = 0;
 if (n <= 3) return A;

 vector<pair<int, int>> ans(2 * n);

 sort(A.begin(), A.end());

 for (int i = 0; i < n; ++i) {
  while (k >= 2 && cross_product(ans[k - 2], ans[k - 1], A[i
      ]) <= 0) {
   k--;
  }
  ans[k++] = A[i];
 }

 for (int i = n - 1, t = k + 1; i > 0; --i) {
  while (k >= t && cross_product(ans[k - 2], ans[k - 1], A[i
      - 1]) <= 0) {
   k--;
  }
  ans[k++] = A[i - 1];
 }

 ans.resize(k - 1);

 return ans;
}
```

6 Graph Theory

6.1 0-1 BFS [SA]

```
const int INF = 10000;
void bfs(vector<vector<pair<int, int>>>& adj, int n, int s)
    {
    vector<int> dist(n, INF);
    dist[s] = 0;
    deque<int> q;
    q.push_front(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop_front();
        for (auto edge : adj[u]) {
            int v = edge.first;
            int w = edge.second;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                if (w == 1) {
                    q.push_back(v);
                }
                else {
                    q.push_front(v);
                }
            }
        }
    }
}
```

6.2 Articulation Points [SA]

```
const int N = 100001;
vector<int> adj[N];
bool visited[N], is_ap[N];
int tin[N], low[N];

void dfs(int u, int par, int& time) {
    int children = 0;
```

```
    visited[u] = true;
    tin[u] = low[u] = ++time;

    for (auto v : adj[u]) {
        if (!visited[v]) {
            ++children;
            dfs(v, u, time);
            low[u] = min(low[u], low[v]);
            if (par != -1 && low[v] >= tin[u]) {
                is_ap[u] = true;
            }
        }
        else if (v != par) {
            low[u] = min(low[u], tin[v]);
        }
    }
    if (par == -1 && children > 1) {
        is_ap[u] = true;
    }
}
vector<int> get_ap(int n) {
    int time = 0;
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i, -1, time);
    }
    vector<int> ap;
    for (int i = 0; i < n; ++i) {
        if (is_ap[i]) {
            ap.push_back(i);
        }
    }
    return ap;
}
```

6.3 Breadth First Search [SA]

```
const int N = 10001;
int dist[N], parent[N];
vector<int> adj[N];

void init() {
    memset(dist, -1, sizeof dist);
    memset(parent, -1, sizeof parent);
}
}

void bfs(int s) {
    queue<int> q;
    q.push(s);
    dist[s] = 0, parent[s] = -1;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : adj[u]) {
            if (dist[v] == -1) {
                dist[v] = dist[u] + 1;
                parent[v] = u;
                q.push(v);
            }
        }
    }
}
```

6.4 Bridges [SA]

```
const int N = 100001;
vector<int> adj[N];
bool visited[N];
int tin[N], low[N];

void dfs(int u, int p, int& timer) {
    visited[u] = true;
    tin[u] = low[u] = ++timer;
    for (int v : adj[u]) {
        if (v == p) continue;
        if (visited[v]) {
            low[u] = min(low[u], tin[v]);
        }
        else {
            dfs(v, u, timer);
            low[u] = min(low[u], low[v]);
            if (low[v] > tin[u]) {
                // u-v is a bridge
            }
        }
    }
}
void find_bridges(int n) {
    int timer = 0;
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i, -1, timer);
    }
}
```

6.5 Depth First Search [SA]

```
const int N = 10001;
bool visited[N];
vector<int> adj[N];

void dfs(int u) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v);
        }
    }
}
```

6.6 Dijkstra's Algorithm [SA]

```
const int N = 100001;
struct Node {
    int vertex, weight;
};

int dist[N], parent[N];
bool vis[N];
vector<Node> adj[N];

void init() {
    for (int i = 0; i < N; ++i) {
        dist[i] = INT_MAX, parent[i] = -1, vis[i] = false;
    }
}

struct cmp {
    bool operator()(Node& a, Node& b) const {
        return a.weight > b.weight;
    }
};

void dijkstra(int s) {
    dist[s] = 0, vis[s] = true;

    priority_queue<Node, vector<Node>, cmp> PQ;
    PQ.push({s, 0});

    while (!PQ.empty()) {
        Node cur = PQ.top();
        PQ.pop();
        int u = cur.vertex;
        for (auto&& [v, w] : adj[u]) {
```

```
            if (!vis[v] && (dist[u] + w) < dist[v]) {
                dist[v] = dist[u] + w;
                parent[v] = u;
                PQ.push({v, dist[v]});
            }
        }
        vis[u] = true;
    }
}
```

6.7 Floyd Warshall Algorithm

```
const int INF = 999999;

void floyd_warshall(vector<vector<int>>& d, int n) {
    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                if (d[i][k] < INF && d[k][j] < INF)
                    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
```

6.8 Lowest Common Ancestor [SA]

```
const int N = 100001;
const int LG = 1 + __lg(N);
auto dist = vector(N + 1, INT_MAX);
auto LCA = vector(N + 1, vector(LG + 1, -1));
vector<vector<int>> adj;

void dfs(int s, int p) {
    LCA[s][0] = p;
    for (auto i : adj[s]) {
        if (dist[i] == INT_MAX) {
            dist[i] = 1 + dist[s];
            dfs(i, s);
        }
    }
}

void preprocess(int s) {
    dist[s] = 0;
    dfs(s, -1);
    for (int i = 1; i <= LG; ++i) {
        for (int j = 0; j <= N; ++j) {
            int p = LCA[j][i - 1];
            if (p == -1) continue;
            LCA[j][i] = LCA[p][i - 1];
        }
    }
}

int get_lca(int u, int v) {
    if (dist[u] > dist[v])
        swap(u, v);

    int dif = dist[v] - dist[u];
    while (dif > 0) {
        int lg = __lg(dif);
        v = LCA[v][lg];
        dif -= (1 << lg);
    }
    if (u == v)
        return u;

    for (int i = LG; i >= 0; --i) {
        if (LCA[u][i] == -1 || LCA[u][i] == LCA[v][i])
            continue;
        u = LCA[u][i];
        v = LCA[v][i];
    }
    return LCA[u][0];
}
```

6.9 Topological Sort [SA]

```
const int N = 10001;
bool visited[N];
vector<int> seq;

void dfs(vector<vector<int>>& adj, int u) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v])
            dfs(adj, v);
    }
    seq.push_back(u);
}

void topological_sort(vector<vector<int>>& adj, int nodes) {
    for (int i = 1; i <= nodes; ++i) {
        if (!visited[i]) dfs(adj, i);
    }
    reverse(seq.begin(), seq.end());
}
```

7 Mathematics

7.1 Collinear Check [SA]

```
bool collinear(int x1, int y1, int x2, int y2, int x3, int
    y3) {
    return (y2 - y1) * (x3 - x2) == (x2 - x1) * (y3 - y2);
}
```

7.2 Mathematical Progression [SA]

```
int arithmetic_nth_term(int a, int n, int d) {
    return a + (n - 1) * d;
}
int arithmetic_sum(int a, int n, int d) {
    return n * (2 * a + (n - 1) * d) / 2;
}
int geometric_nth_term(int a, int n, int r) {
    return a * pow(r, n - 1);
}
int geometric_sum(int a, int n, int r) {
    if (r == 1) return n * a;
    if (r < 1) return a * (1 - pow(r, n)) / (1 - r);
    else return a * (pow(r, n) - 1) / (r - 1);
}
int infinite_geometric_sum(int a, int r) {
    assert(r < 1);
    return a / (1 - r);
}
```

8 Misc

8.1 128 Bit Integer Utility [SA]

```
namespace int128_utility {
    std::istream &operator>>(std::istream &in, __int128 &n) {
        std::string s;
        in >> s;
        bool neg = !s.empty() && s.front() == '-';
        n = 0;
```

```
        for (size_t i = 0 + neg; i < s.length(); ++i) {
            n = n * 10 + (s[i] - '0');
        }
        if (neg) n *= -1;
        return in;
    }
    std::ostream &operator<<(std::ostream &out, __int128 n) {
        bool neg = n < 0;
        std::string s;
        do {
            s += to_string(abs(int(n % 10)));
            n /= 10;
        } while (neg ? n < 0 : n > 0);
        if (neg) s += '-';
        std::reverse(s.begin(), s.end());
        out << s;
        return out;
    }
}
using namespace int128_utility;
```

8.2 Bit Manipulation [RH]

```
long long int get_bit(long long int n, long long int pos)
{
    return (n & (1 << pos));
}
long long int set_bit(long long int n, long long int pos)
{
    return (n | (1 << pos));
}
long long int clear_bit(long long int n, long long int pos)
{
    return (n & (~(1 << pos)));
}
long long int updater_bit(long long int n, long long int pos
    , long long int value)
{
    n = n & (~(1 << pos));
    return (n | (1 << value));
}
```

8.3 Custom Hash Function [SA]

```
struct chash {
    const uint64_t C = uint64_t(2e18 * numbers::pi) + 71;
    const unsigned int RANDOM = chrono::high_resolution_clock
        ::now().time_since_epoch().count();

    uint64_t operator()(int64_t x) const {
        return __builtin_bswap64((x ^ RANDOM) * C);
    }
};
```

8.4 Directional Array [SA]

```
const int dx[] = {-1, +1, +0, +0, -1, -1, +1, +1};
const int dy[] = {+0, +0, -1, +1, -1, +1, -1, +1};
```

9 Number Theory

9.1 Binary Exponentiation [SA]

```
int bin_expo(int b, int e, int m) {
    b %= m;
    int res = 1;
    while (e > 0) {
        if (e & 1) res = b * res % m;
        b = b * b % m;
        e >>= 1;
    }
    return res;
}
```

9.2 Euler's Totient Function [SA]

```
vector<int> phi;
void totient(int n) {
    phi.resize(n + 1);
    phi[0] = 0, phi[1] = 1;
    for (int i = 2; i <= n; ++i) {
        phi[i] = i;
    }

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

9.3 Extended Euclidean Algorithm [SA]

```
int egcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int gcd = egcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return gcd;
}
```

9.4 Factorial Trailing Zeros [SA]

```
int64_t fact_trailing_zeros(int64_t n) {
    int64_t count = 0;
    for (int64_t i = 5; i <= n; i *= 5)
        count += (n / i);
    return count;
}
```

9.5 Fraction Functions [SK]

```
pair<ll, ll> frac_add(pair<ll, ll> a, pair<ll, ll> b)
{
    ll g = a.second * b.second;
    pair<ll, ll> x;
    x.second = g;
    x.first = a.first * (b.second) + b.first * (a.second);
    ll y = __gcd(x.first, x.second);
    x.first /= y;
    x.second /= y;
    return x;
}
pair<ll, ll> frac_mult(pair<ll, ll> a, pair<ll, ll> b)
{

    pair<ll, ll> x;
```

```
    x.first = a.first * b.first;
    x.second = a.second * b.second;
    ll y = __gcd(x.first, x.second);
    x.first /= y;
    x.second /= y;
    return x;
}
```

9.6 Integer Factorization [SA]

```
vector<int64_t> factorization(int64_t n) {
    vector<int64_t> pf;
    while (n % 2 == 0) pf.push_back(2), n /= 2;

    for (int64_t i = 3; i * i <= n; i += 2)
        while (n % i == 0) pf.push_back(i), n /= i;

    if (n > 1) pf.push_back(n);
    return pf;
}
```

9.7 Miller Rabin Primality Test [SK]

```
ll mulmod(ll a, ll b, ll c) {
    ll x = 0, y = a % c;
    while (b) {
        if (b & 1) x = (x + y) % c;
        y = (y << 1) % c;
        b >>= 1;
    }
    return x % c;
}

ll fastPow(ll x, ll n, ll MOD) {
    ll ret = 1;
    while (n) {
        if (n & 1) ret = mulmod(ret, x, MOD);
        x = mulmod(x, x, MOD);
        n >>= 1;
    }
    return ret;
}

bool isPrime(ll n) {
    ll d = n - 1;
    int s = 0;
    while (d % 2 == 0) {
        s++;
        d >>= 1;
    }

    // It's guranteed that these values will work for any
        number smaller than 3*10**18 (3 and 18 zeros)
    int a[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (int i = 0; i < 9; i++) {
        bool comp = fastPow(a[i], d, n) != 1;
        if (comp) for (int j = 0; j < s; j++) {
            ll fp = fastPow(a[i], (1LL << (ll) j) * d, n);
            if (fp == n - 1) {
                comp = false;
                break;
            }
        }
        if (comp) return false;
    }
    return true;
}
```

9.8 Modular Inverse 1 [SA]

```
int invModF(int a, int m) {
    // Modular multiplicative inverse using fermat's little
        theorem
    // m is a prime number
    return bin_expo(a, m - 2, m);
}
```

9.9 Modular Inverse 2 [SA]

```
int egcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    int x1, y1;
    int gcd = egcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return gcd;
}

int invModE(int a, int m) {
    // Modular multiplicative inverse using extended
        euclidean algorithm
    // a and m are coprime
    int x, y, g = egcd(a, m, x, y);
    if (x < 0)
        x = (x + m) % m;

    return x;
}
```

9.10 Mul Mod [SA]

```
int mul_mod(int a, int b, int mod) {
    int x = 0, y = a % mod;
    while (b > 0) {
        if (b & 1) {
            x = (x + y) % mod;
        }
        y = (y * 2) % mod;
        b /= 2;
    }
    return x % mod;
}
```

9.11 Pollard's Rho Algorithm [SK]

```
ll mul(ll x, ll y, ll mod) {
    ll res = 0;
    x %= mod;
    while (y) {
        if (y & 1) res = (res + x) % mod;
        y >>= 1;
        x = (x + x) % mod;
    }
    return res;
}
ll bigmod(ll a, ll m, ll mod) {
    a = a % mod;
    ll res = 1ll;
    while (m > 0) {
        if (m & 1) res = mul(res, a, mod);
        m >>= 1;
        a = mul(a, a, mod);
    }
    return res;
}
bool composite(ll n, ll a, ll s, ll d) {
    ll x = bigmod(a, d, n);
    if (x == 1 or x == n - 1) return false;
```

```
    for (int r = 1; r < s; r++) {
        x = mul(x, x, n);
        if (x == n - 1) return false;
    }
    return true;
}
bool isprime(ll n) {
    if (n < 4) return n == 2 or n == 3;
    if (n % 2 == 0) return false;
    ll d = n - 1;
    ll s = 0;
    while (d % 2 == 0) {
        d /= 2;
        s++;
    }
    for (int i = 0; i < 10; i++) {
        ll a = 2 + rand() % (n - 3);
        if (composite(n, a, s, d)) return false;
    }
    return true;
}
// Polard rho
ll f(ll x, ll c, ll mod) {
    return (mul(x, x, mod) + c) % mod;
}
ll rho(ll n) {
    if (n % 2 == 0) {
        return 2;
    }
    ll x = rand() % n + 1;
    ll y = x;
    ll c = rand() % n + 1;
    ll g = 1;
    while (g == 1) {
        x = f(x, c, n);
        y = f(y, c, n);
        y = f(y, c, n);
        g = __gcd(abs(y - x), n);
    }
    return g;
}
void factorize(ll n, vector<ll>& factors) {
    if (n == 1) {
        return;
    }
    else if (isprime(n)) {
        factors.push_back(n);
        return;
    }
    ll cur = n;
    for (ll c = 1; cur == n; c++) {
        cur = rho(n);
    }
    factorize(cur, factors), factorize(n / cur, factors);
}
```

9.12 Segmented Sieve [SA]

```
vector<bool> segmented_sieve(int64_t L, int64_t R) {
    int64_t LIMIT = 1 + sqrt(R);
    vector<bool> mark(LIMIT);
    vector<int64_t> primes;
    for (int64_t i = 2; i < LIMIT; ++i) {
        if (!mark[i]) {
            primes.push_back(i);
            for (int64_t j = i * i; j < LIMIT; j += i)
                mark[j] = true;
        }
    }
    vector<bool> is_prime(R - L + 1, true);
    for (auto& i : primes)
        for (int64_t j = max(i * i, (L + i - 1) / i * i); j
             <= R; j += i)
            is_prime[j - L] = false;
    is_prime[0] = L != 1;
    return is_prime;
}
```

9.13 Sieve of Eratosthenes [SA]

```
const int MX_N = 1e8;
bitset<MX_N + 1> isPrime;

void sieve() {
    // false = prime, true = not prime
    isPrime[0] = isPrime[1] = true;
    for (int i = 4; i <= MX_N; i += 2) isPrime[i] = true;

    for (int64_t i = 3; i * i <= MX_N; i += 2) {
        if (!isPrime[i]) {
            for (int64_t j = i * i; j <= MX_N; j += i + i)
                isPrime[j] = true;
        }
    }
}
```

9.14 Sieve [RH]

```
vector <bool> isprime(1000050, true);
vector <int> primelist;
void sieve()
{
    isprime[1] = false;
    isprime[0] = false;
    isprime[2] = true;
    for (int i = 4; i <= 1000050; i += 2)
    {
        isprime[i] = false;
    }
    for (long long int i = 3; i * i <= 1000050; i++)
    {
        for (long long int j = i * i; j <= 1000050; j = j + i
             + i)
        {
            isprime[j] = false;
        }
    }
    for (int i = 1; i <= 1000050; i++)
    {
        if (isprime[i])
        {
            primelist.push_back(i);
        }
    }
}
```

9.15 Sum of Divisors [SA]

```
const int N = 1e6;
int SOD[N + 1];
void generate_sod() {
    for (int i = 1; i <= N; ++i) {
        for (int j = i; j <= N; j += i) {
            SOD[j] += i;
        }
    }
}
```

10 Strings

10.1 Fast Pattern Matching [SA]

```
vector<int> get_matches(string s, string p) {
    vector<int> res;
    for (int i = s.find(p); i != string::npos; i = s.find(p,
         i + 1)) {
        res.push_back(i);
    }
    return res;
}
```

10.2 KMP Algorithm [SA]

```
vector<int> get_lps(const string& p) {
    int n = p.length();
    vector<int> lps(n);

    for (int i = 1, j = 0; i < n;) {
        if (p[i] == p[j]) {
            lps[i] = j + 1;
            ++j, ++i;
        }
        else if (j != 0) {
            j = lps[j - 1];
        }
        else {
            lps[i] = j, ++i;
        }
    }
    return lps;
}
vector<int> kmp(string s, string p) {
    vector<int> lps = get_lps(p), res;
    int n = s.length();
    int m = p.length();

    for (int i = 0, j = 0; i < n;) {
        if (s[i] == p[j]) {
            ++i, ++j;
        }
        else if (j != 0) {
            j = lps[j - 1];
        }
        else {
            ++i;
        }
        if (j == m) {
            res.push_back(i - m);
            j = lps[j - 1];
        }
    }
    return res;
}
```

10.3 Rabin Karp Algorithm [SA]

```
const int base = 347, mod = 1000000007;

int get_hash(const string& s, int m) {
    int hash = 0, exp = 1;
    for (int i = m - 1; i >= 0; --i) {
        hash = (hash % mod + (s[i] * exp) % mod) % mod;
        exp = (exp * base) % mod;
    }
    return hash;
}
vector<int> rabin_karp(const string& s, const string& p) {
    vector<int> pos;
    int n = s.length(), m = p.length();
    if (n < m || m == 0 || n == 0)
        return pos;

    int exp = 1;
    for (int i = 1; i <= m - 1; ++i) {
        exp = (exp * base) % mod;
    }

    int h_txt = get_hash(s, m);
    int h_pat = get_hash(p, m);

    if (h_txt == h_pat) {
        pos.push_back(0);
    }
    for (int i = m; i < n; ++i) {
        h_txt = (h_txt - (exp * s[i - m]) % mod) % mod;
        h_txt = (h_txt + mod) % mod;
        h_txt = (h_txt * base) % mod;
        h_txt = (h_txt + s[i]) % mod;

        if (h_txt == h_pat) {
            pos.push_back(i - m + 1);
        }
    }
    return pos;
}
```

10.4 String Division [SA]

```
int MOD(string s, int n) {
    int rem = 0;
    for (char i : s) {
        int dig = i - '0';
        rem = (rem * 10) + dig;
        rem %= n;
    }
    return rem;
}
```

10.5 Z Algorithm [SA]

```
vector<int> z_function(string s) {
    int n = s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; 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;
        }
    }
    return z;
}
```

11 Templates

11.1 Riasad Huq

```
#include<bits/stdc++.h>
using namespace std;
#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))
#define endl "\n"
#define ios ios_base::sync_with_stdio(false);cin.tie(NULL);
#include<string>
#include<string.h>

int main()
{
```

```
    ios;
    return 0;
}
```

11.2 Sarwar Alam

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
using namespace std;
template<typename T>
using ordered_set = tree<T, null_type, less <T>, rb_tree_tag
     , tree_order_statistics_node_update>;
#define dbg(x) " ["#x << ": " << x << "] "

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    return 0;
}
```

11.3 Sarwar Khalid

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

typedef long long ll;
typedef unsigned long long ull;
#define endl "\n"
#define pi 3.142
const double eps = 1e-10;
int dx[] = {1, 0, -1, 0};
int dy[] = {0, 1, 0, -1};

const ll M = (ll) (1e9) + 7;
const ll inf = (ll) 1e17;
const int N = (ll) (1e6 + 10);

int main()
{
    cin.tie(0);
    cout.tie(0);
    ios_base::sync_with_stdio(false);

    //freopen("two.in", "r", stdin);
    //freopen("out.txt", "w", stdout);
}

/*

*/
```