Contents

Data structures 3
Ordered set 3
Disjoint set union 3
DSU 3
DSU bipartiteness 4
DSU apps 5
Segment tree 6
Segment tree 6
Segment tree max sum range 7
Sparse table 8
LCA 8
Range Minimum Query 9
Binary search tree 10
BST 10
AVL 12
Fenwick Tree (BIT) 13
MO algorithm 15
SQRT Decomposition 15
SQRT Decomposition 15
Example 16
Big Intger 18
Graphs 19
shortest path algorithms 19
Dijkstra 19
Bellmanford 20
Difference constraints 20
Floyd 21
SPFA (shortest path faster algorithm) 21
Tarjan 23
Strongly connected component 23
Articulation point and bridge 24
Edge Classification 24
Kruskal (minimum spanning tree) 25
2 SAT 25

Maximum bipartite matching 26
Geometry 26
Points 26
Lines 29
Triangles 32
Circles 34
Math's 36
Elementary 36
Primes 37
Prime Factorization 38
Factorization 40
Mod Inverse 40
Combinatorics 41
Matrices 42
string processing 44
KMP 44
Trie 45
Suffix array 46
LIS binary Search 47
Bitmask 48
Sort 48
Merge sort 48
Radix sort 49
Coordinate Compress 49
Hash 49
Random number 50
Java 50
Scanner 50
Segment tree 51
Leap Year 53

Data structures

Ordered set

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
template<typename key>
using ordered_set = tree<key, null_type, less<key>, rb_tree_tag,
tree_order_statistics_node_update>;
/*
find_by_order(k) :
It returns to an iterator to the k-th element (counting from zero) in the set in
O(logn) time.
To find the first element k must be zero.
order_of_key(k) :
It returns to the number of items that are strictly smaller than our item k in
O(logn) time.
*/
```

Disjoint set union

DSU

```
struct DSU {
	vector<int> rank, parent, size;
	vector<vector<int>> component;
	int    forsets;
	DSU(int n) {
		size = rank = parent = vector<int>(n + 1, 1);
		component = vector<vector<int>>(n + 1);
		forsets = n;
		for (int i = 0; i <= n; i++) {
			parent[i] = i;
			component[i].push_back(i);
		}
	}
	int find_set(int v) {
		if (v == parent[v])return v;
		return parent[v] = find_set(parent[v]);
	}
	void link(int par, int node) {
		parent[node] = par;
		size[par] += size[node];
		for (const int& it : component[node])
			component[par].push_back(it);
		component[node].clear();
		if (rank[par] == rank[node])rank[par]++;
		forsets--;
	}
	bool union_sets(int v, int u) {
		v = find_set(v), u = find_set(u);
```

```
		if (v != u) {
			if (rank[v] < rank[u])swap(v, u);
			link(v, u);
		}
		return v != u;
	}
	bool same_set(int v, int u) {
		return find_set(v) == find_set(u);
	}
	int size_set(int v) {
		return size[find_set(v)];
	}
};
```

DSU bipartiteness

```
struct DSU_bipartiteness {
	vector<int> bipartite, rank;
	vector<pair<int, int>> parent;
	DSU_bipartiteness(int n) {
		bipartite = rank = vector<int>(n + 1, 1);
		parent = vector<pair<int, int>>(n + 1);
		for (int i = 0; i <= n; i++)
			parent[i] = { i,0 };
	}
	pair<int, int> find_set(int x) {
		if (x == parent[x].first)return parent[x];
		int parity = parent[x].second;
		parent[x] = find_set(parent[x].first);
		parent[x].second ^= parity;
		return parent[x];
	}
	void union_sets(int x, int y) {
		pair<int, int> p = find_set(x);
		x = p.first;
		int paX = p.second;
		p = find_set(y);
		y = p.first;
		int paY = p.second;
		if (x == y) {
			if (paX == paY)
				bipartite[x] = false;
		}
		else {
			if (rank[x] < rank[y]) swap(x, y);
			parent[y] = { x, paX ^ paY ^ 1 };
			bipartite[x] &= bipartite[y];
			if (rank[x] == rank[y]) rank[x]++;}}
```

```
		bool is_bipartite(int x) {
			return bipartite[find_set(x).first];
		}
};
```

DSU apps

```
#include"DSU.h"

void Painting_subarrays() {
	struct Query {
		int l, r, c;
		Query(int l, int r, int c) :l(l), r(r), c(c) {}
	};
	int n, q; cin >> n >> q;
	DSU uf(n);
	vector<int> ans(n + 1);
	vector<Query> query(q);
	for (int i = 0; i < q; i++)cin >> query[i].l >> query[i].r >> query[i].c;
	reverse(query.begin(), query.end());
	for (auto q : query) {
		int l = q.l, r = q.r, c = q.c;
		for (int cur = uf.find_set(l); cur <= r; cur = uf.find_set(cur)) {
			uf.parent[cur] = cur + 1;
			ans[cur] = c;
		}
	}
}
void RMQ() {
	struct Query {
		int l, r, idx;
		Query(int l, int r, int idx) :l(l), r(r), idx(idx) {}
	};
	int n, q;
	cin >> n >> q;
	vector<int> v(n);
	vector<vector<Query>> query(n);
	vector<int> ans(q);
	DSU uf(n);
	for (auto& a : v)cin >> a;
	for (int i = 0; i < q; i++) {
		int l, r;
		cin >> l >> r;
		query[r].push_back(Query(l, r, i));
	}
	stack<int> st;
	for (int i = 0; i < n; i++) {
		while (!st.empty() && v[st.top()] > v[i]) {
			uf.parent[st.top()] = i;
			st.pop();}
```

```
			st.push(i);
			for (auto q : query[i])
				ans[q.idx] = v[uf.find_set(q.l)];
		}
}
```

Segment tree

Segment tree

```
template<class node, class treeType, class lazyType>
struct segment_tree {
	int n;
	vector<node> arr;
	vector<treeType> tree;
	vector<lazyType> lazy;
	segment_tree(int n) {
		this->n = n;
		tree = vector<treeType>(n << 2);
		lazy = vector<lazyType>(n << 2);
	}
	segment_tree(vector<node>& _arr) {
		n = _arr.size() - 1;
		tree = vector<treeType>(n << 2);
		lazy = vector<lazyType>(n << 2);
		arr = _arr;
		build(1, 1, n);
	}
	void update(int from, int to, lazyType val) {
		update(1, 1, n, from, to, val);
	}
	node query(int from, int to) {
		return query(1, 1, n, from, to);
	}
	treeType merge(treeType a, treeType b);
	void build(int idx, int start, int end) {
		if (start == end) {
			tree[idx] = arr[start]; return;
		}
		int mid = start + end >> 1;
		build(idx << 1, start, mid);
		build(idx << 1 | 1, mid + 1, end);
		tree[idx] = merge(tree[idx << 1], tree[idx << 1 | 1]);
	}
	void propagate(int idx, int start, int end) {
		if (lazy[idx] == 0) return;
		tree[idx] += lazy[idx];
		if (start != end) {
			lazy[idx << 1] += lazy[idx];
			lazy[idx << 1 | 1] += lazy[idx];
		}
		lazy[idx] = 0;
	}
```

```
	void update(int idx, int start, int end, int from, int to, lazyType val) {
		propagate(idx, start, end);
		if (to < start || end < from)return;
		if (from <= start && end <= to) {
			lazy[idx] += val;
			propagate(idx, start, end);
			return;
		}
		int mid = start + end >> 1;
		update(idx << 1, start, mid, from, to, val);
		update(idx << 1 | 1, mid + 1, end, from, to, val);
		tree[idx] = merge(tree[idx << 1], tree[idx << 1 | 1]);
	}
	treeType query(int idx, int start, int end, int from, int to) {
		propagate(idx, start, end);
		if (from <= start && end <= to)return tree[idx];
		int mid = start + end >> 1;
		if (to <= mid)
			return query(idx << 1, start, mid, from, to);
		else if (mid < from)
			return query(idx << 1 | 1, mid + 1, end, from, to);
		treeType a = query(idx << 1, start, mid, from, to);
		treeType b = query(idx << 1 | 1, mid + 1, end, from, to);
		return merge(a, b);
	}
};
```

Segment tree max sum range

```
#define ll long long

struct node {
	ll left, right, mid, sum;
	node(ll val = 0) { left = right = mid = sum = val; }
	ll getMax() {
		return max({ left,right,mid,sum });
	}
};

struct segment_tree {
	int n;
	vector<node> tree, arr;
	segment_tree(int n = 0) :n(n) {
		arr = vector<node>(n + 1);
		tree = vector<node>(n << 2);
	}
	segment_tree(vector<node>& _arr) {
		n = _arr.size() - 1;
		tree = vector<node>(n << 2);
		arr = _arr;
		build(1, 1, n);
	}
	void update(int pos, int val) {
		update(1, 1, n, pos, val);
	}
```

```
	ll query(int from, int to) {
		return query(1, 1, n, from, to).getMax();
	}
	node merge(node a, node b) {
		node c;
		c.left = max(a.left, a.sum + b.left);
		c.right = max(b.right, b.sum + a.right);
		c.mid = max({ a.mid,b.mid,a.right + b.left });
		c.sum = a.sum + b.sum;
		return c;
	}
	void build(int idx, int start, int end) {
		if (start == end) {
			tree[idx] = arr[start]; return;
		}
		int mid = start + end >> 1;
		build(idx << 1, start, mid);
		build(idx << 1 | 1, mid + 1, end);
		tree[idx] = merge(tree[idx << 1], tree[idx << 1 | 1]);
	}
	void update(int idx, int start, int end, int pos, int val) {
		if (start == end) {
			tree[idx] = val; return;
		}
		int mid = start + end >> 1;
		if (pos <= mid)
			update(idx << 1, start, mid, pos, val);
		else
			update(idx << 1 | 1, mid + 1, end, pos, val);
		tree[idx] = merge(tree[idx << 1], tree[idx << 1 | 1]);
	}
	node query(int idx, int start, int end, int from, int to) {
		if (from <= start && end <= to)return tree[idx];
		int mid = start + end >> 1;
		if (to <= mid)
			return query(idx << 1, start, mid, from, to);
		else if (mid < from)
			return query(idx << 1 | 1, mid + 1, end, from, to);
		node a = query(idx << 1, start, mid, from, to);
		node b = query(idx << 1 | 1, mid + 1, end, from, to);
		return merge(a, b);
	}
};
```

Sparse table

LCA

```
int logN;
vector<int> depth;
vector<vector<int>> adj, lca;
void dfs(int node = 1, int parent = -1) {
	lca[node][0] = parent;
	if (~parent)depth[node] = depth[parent] + 1;
```

```
	for (int child : adj[node])if (child != parent)
		dfs(child, node);
}


// return first = lca,second = distance between the two nodes
pair<int, int> LCA(int u, int v) {
	if (depth[u] < depth[v])
		swap(u, v);
	int dis = 0;
	for (int k = logN; k >= 0; k--)
		if (depth[u] - (1 << k) >= depth[v])
			u = lca[u][k], dis += (1 << k);
	if (u == v)return { u,dis };
	for (int k = logN; k >= 0; k--) {
		if (lca[u][k] != lca[v][k]) {
			u = lca[u][k];
			v = lca[v][k];
			dis += (1 << k + 1);
		}
	}
	return { lca[u][0],dis + 2 };
}
void build() {
	int n;
	cin >> n;
	logN = log2(n);
	adj = vector<vector<int>>(n + 1);
	lca = vector<vector<int>>(n + 1, vector<int>(logN + 1, -1));
	depth = vector<int>(n + 1);
	for (int i = 1; i < n; i++) {
		int u, v;
		cin >> u >> v;
		adj[u].push_back(v);
		adj[v].push_back(u);
	}
	dfs();
	for (int k = 1; k <= logN; k++)
		for (int node = 1; node <= n; node++) {
			int parent = lca[node][k - 1];
			if (parent != -1)
				lca[node][k] = lca[parent][k - 1];
		}
}
```

Range Minimum Query

```
vector<int> v, lg;
vector<vector<int>> sparseTable;
```

```
bool isPowerOfTwo(int num) { return (num & num - 1) == 0; }
int Min(int idx1, int idx2) {
	return (v[idx1] <= v[idx2] ? idx1 : idx2);
}
// O(n * log(n))
void buildSparesTable() {
	int n = v.size();
	lg = vector<int>(n + 1); // to get log2 in O(1)
	for (int i = 2; i <= n; i++) {
		lg[i] = lg[i - 1];
		if (isPowerOfTwo(i)) lg[i]++;
	}
	int logN = lg[n];
	sparseTable = vector<vector<int>>(n, vector<int>(logN + 1));
	for (int i = 0; i < n; i++) sparseTable[i][0] = i;
	for (int k = 1; k <= logN; k++)
		for (int i = 0; i + (1 << k - 1) < n; i++) {
			sparseTable[i][k] = Min(sparseTable[i][k - 1],
				sparseTable[i + (1 << k - 1)][k - 1]);
		}
}
// O(1)
int rangeMinimumQuery(int l, int r) {
	int k = lg[r - l + 1];// max k ==> 2^k <= lenth of range
	//check first 2^k from left ans last 2^k from right //overlap
	return Min(sparseTable[l][k], sparseTable[r - (1 << k) + 1][k]);
}
```

Binary search tree

BST

```
struct node {
	int key;
	node* left, * right, * parent;
	node() { key = 0; left = right = parent = NULL; }
	node(int key, node* left = NULL, node* right = NULL, node* parent = NULL) :
		key(key), left(left), right(right), parent(parent) {}
};
typedef node* nodeptr;
class BST {
public:
	nodeptr root;
	BST() : root(NULL) {}
	nodeptr find(int key) { return find(root, key); }
	void insert(int key) { root = insert(root, key); }
	void erase(int key) { root = erase(root, key); }
	nodeptr minimum(nodeptr root) {
		if (root->left == NULL)return root;
		return minimum(root->left);
	}
	nodeptr maximum(nodeptr root) {
		if (root->right == NULL)return root;
```

```
	return maximum(root->right);
}
nodeptr successor(nodeptr cur) {//smallest key larger than cur
	if (cur->right != NULL)return minimum(cur->right);
	nodeptr tmp = cur->parent;
	while (tmp != NULL && tmp->right == cur)
		cur = tmp, tmp = tmp->parent;
	return tmp;
}
nodeptr bredecessor(nodeptr cur) {//biggest key less than cur
	if (cur->left != NULL)return maximum(cur->left);
	nodeptr tmp = cur->parent;
	while (tmp != NULL && tmp->left == cur)
		cur = tmp, tmp = tmp->parent;
	return tmp;
}
nodeptr find(nodeptr root, int key) {
	if (root == NULL)return NULL;
	if (key == root->key)return root;
	if (key < root->key)return find(root->left, key);
	return find(root->right, key);
}
nodeptr insert(nodeptr root, int key) {
	if (root == NULL)root = new node(key);
	else if (key < root->key) {
		root->left = insert(root->left, key);
		root->left->parent = root;
	}
	else if (key > root->key) {
		root->right = insert(root->right, key);
		root->right->parent = root;
	}
	return root;
}
nodeptr erase(nodeptr root, int key) {
	if (root == NULL)return root;
	if (key < root->key) {
		root->left = erase(root->left, key);
		root->left->parent = root;
	}
	else if (key > root->key) {
		root->right = erase(root->right, key);
		root->right->parent = root;
	}
	else {
		nodeptr tmp;

		if (root->left == NULL || root->right == NULL) {
			if (root->left == NULL)tmp = root->right;
			else tmp = root->left;
			free(root);
			return tmp;
		}
		else {
			tmp = successor(root);
```

```
                        root->key = tmp->key;
                        root->right = erase(root->right, tmp->key);
                        root->right->parent = root;
                    }
                }
                return root;
        }
};
void inorder(nodeptr root) {
        if (root == NULL)return;
        inorder(root->left);
        cout << root->key << ' ';
        inorder(root->right);
}
void preorder(nodeptr root) {
        if (root == NULL)return;
        cout << root->key << ' ';
        preorder(root->left);
        preorder(root->right);
}
void postorder(nodeptr root) {
        if (root == NULL)return;
        postorder(root->left);
        postorder(root->right);
        cout << root->key << ' ';
}
```

AVL

```
#include"BST.h"
struct AVLnode {
        int key, height;
        AVLnode* left, * right, * parent;
        static AVLnode* sentinel;
        AVLnode() {
                parent = left = right = sentinel;
                height = 0;
        }
        AVLnode(int key) : key(key) {
                parent = left = right = sentinel;
                height = 0;
        }
        void updateHeight() {
                height = 1 + max(left->height, right->height);
        }
        int balanceFactor() {
                return left->height - right->height;
        }
};

AVLnode* AVLnode::sentinel = new AVLnode();
class AVL : public BST {
        typedef AVLnode* nodeptr;
public:
        nodeptr root;
        AVL() : root(NULL) {}
        void insert(int key) { root = insert(root, key); }
private:
```

```
	nodeptr rightRotation(nodeptr Q) {
		nodeptr P = Q->left;
		Q->left = P->right;
		Q->left->parent = Q;
		P->right = Q;
		P->parent = Q->parent;
		Q->parent = P;
		Q->updateHeight();
		P->updateHeight();
		return P;
	}
	nodeptr leftRotation(nodeptr P) {
		nodeptr Q = P->right;
		P->right = Q->left;
		P->right->parent = P;
		Q->left = P;
		Q->parent = P->parent;
		P->parent = Q;
		Q->updateHeight();
		P->updateHeight();
		return Q;
	}
	nodeptr balance(nodeptr root) {
		if (root->balanceFactor() == 2) {
			if (root->left->balanceFactor() == -1)
				root->left = leftRotation(root->left);
			root = rightRotation(root);
		}
		else if (root->balanceFactor() == -2) {
			if (root->right->balanceFactor() == 1)
				root->right = rightRotation(root->right);
			root = leftRotation(root);
		}
		return root;
	}
	nodeptr insert(nodeptr root, int key) {
		if (root == AVLnode::sentinel)
			return root = new AVLnode(key);
		if (key < root->key) {
			root->left = insert(root->left, key);
			root->left->parent = root;
		}
		else if (key > root->key) {
			root->right = insert(root->right, key);
			root->right->parent = root;
		}
		root->updateHeight();
		root = balance(root);
		return root;
	}
};
```

Fenwick Tree (BIT)

```
struct fenwickTree {
	vector<int> BIT;
	int n;
```

```
	fenwickTree(int n) :n(n) {
		BIT = vector<int>(n + 1);
	}
	int getAccum(int idx) {
		int sum = 0;
		while (idx) {
			sum += BIT[idx];
			idx -= (idx & -idx);
		}
		return sum;
	}
	void add(int idx, int val) {
		while (idx <= n) {
			BIT[idx] += val;
			idx += (idx & -idx);
		}
	}
	int getValue(int idx) {
		return getAccum(idx) - getAccum(idx - 1);
	}
	// array must be positive
	int getIdx(int accum) {
		int start = 1, end = (int)BIT.size() - 1, rt = -1;
		while (start <= end) {
			int mid = start + end >> 1;
			int val = getValue(mid);
			if (val >= accum)
				rt = mid, end = mid - 1;
			else start = mid + 1;
		}
		return rt;
	}
};
struct fenwickTree2D {
	vector<vector<int>> BIT;
	void addX(int x, int y, int val) {
		while (x < BIT.size()) {
			addY(x, y, val);
			x += (x & -x);
		}
	}
	void addY(int x, int y, int val) {
		while (y < BIT[x].size()) {
			BIT[x][y] += val;
			y += (y & -y);
		}
	}
};
```

MO algorithm

```
int sqrtN;
struct query {
	int l, r, qIdx, block;
	query(int l, int r, int qIdx) :
		l(l), r(r), qIdx(qIdx), block(l / sqrtN) {}
	bool operator <(const query& o) const {
		if (block != o.block)
			return block < o.block;
		return r < o.r;
	}
};

int curL, curR, ans;
vector<query> q;
vector<int> rt;
void add(int index) {}

void remove(int index) {}

int solve(int l, int r) {
	while (curL > l) add(--curL);
	while (curR < r) add(++curR);
	while (curL < l) remove(curL++);
	while (curR > r) remove(curR--);
	return ans;
}

void MO(int n) {
	sqrtN = sqrt(n);
	rt = vector<int>(q.size());
	ans = curL = curR = 0;
	add(0);
	sort(q.begin(), q.end());
	for (auto it : q)
		rt[it.qIdx] = solve(it.l, it.r);
}
```

SQRT Decomposition

SQRT Decomposition

```
template<typename T, typename Q>
struct node {
	int l, r;
	T lazy;
	node(int l, int r) :l(l), r(r), lazy(0) {}
	void build() {
		//update all bucket using lazy
		//build the bucket
		//clear lazy
	}
	//update all bucket
	void update(T val) {}
	//update range in bucket
	void update(int start, int end, T val) {
		if (start == l && end == r)
```

```
                return update(val);
            //rebuild the bucket if need
        }
        //query about all bucket
        Q query() {}
        //query about range in bucket
        Q query(int start, int end) {
            if (start == l && end == r)
                return query();
            //calc
        }
};
template<typename T, typename Q>
struct SQRT_Decomposition {
        int n, sqrtN;
        vector<node<T, Q>> bucket;
        int begin(int idx) { return idx * sqrtN; }
        int end(int idx) { return min(sqrtN * (idx + 1), n) - 1; }
        int which_block(int idx) { return idx / sqrtN; }
        SQRT_Decomposition(int n) {
            this->n = n;
            sqrtN = sqrt(n);
            for (int i = 0; i < n; i += sqrtN) {
                bucket.push_back(node<T, Q>(i, min(i + sqrtN, n) - 1));
                bucket.back().build();
            }
        }
        void update(int left, int right, T val) {
            int st = which_block(left), ed = which_block(right);
            bucket[st].update(left, min(bucket[st].r, right), val);
            if (st != ed)bucket[ed].update(bucket[ed].l, right, val);
            for (int i = st + 1; i < ed; i++)
                bucket[i].update(val);
        }

        Q query(int left, int right) {
            int st = which_block(left), ed = which_block(right);
            Q rt = bucket[st].query(left, min(bucket[st].r, right));
            if (st != ed)rt += bucket[ed].query(bucket[ed].l, right);
            for (int i = st + 1; i < ed; i++)
                rt += bucket[i].query();
            return rt;
        }
};
```

Example

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define all(v) v.begin(),v.end()
#define sz(v) (int)v.size()
vector<ll> h;
struct node {
        vector<ll> v, sum;
        ll lazy, totalSum;
        int l, r;
        node(int l, int r) :l(l), r(r) {
            lazy = totalSum = 0;}
```

```
        void build() {
                v.clear(); sum.clear();
                for (int i = l; i <= r; i++) {
                        h[i] = max(0LL, h[i] - lazy);
                        v.push_back(h[i]);
                }
                sort(all(v));
                sum.push_back(0);
                for (int i = 0; i < sz(v); i++)
                        sum.push_back(sum.back() + v[i]);
                lazy = 0; totalSum = sum.back();
        }
        void update(ll val) {
                lazy += val;
                int j = upper_bound(all(v), lazy) - v.begin();
                totalSum = sum.back() - sum[j] - (sz(v) - j) * lazy;
        }
        void update(int start, int end, ll val) {
                for (int i = start; i <= end; i++)
                        h[i] = max(0LL, h[i] - val);
                build();
        }
        ll query(int start, int end) {
                ll sum = 0;
                for (int i = start; i <= end; i++)
                        sum += max(h[i] - lazy, 0LL);
                return sum;
        }
};
int n, sqrtN;
vector<node> bucket;
void SQRT_Decomposition() {
        sqrtN = sqrt(n);
        for (int i = 0; i < n; i += sqrtN) {
                bucket.push_back(node(i, min(i + sqrtN - 1, n - 1)));
                bucket.back().build();
        }
}
void update(int left, int right, ll val) {
        int cur = left;
        while (cur <= right) {
                if (cur % sqrtN == 0 && cur + sqrtN - 1 <= right)
                        bucket[cur / sqrtN].update(val), cur += sqrtN;
                else {
                        int endOfBucket = min(right, bucket[cur / sqrtN].r);
                        bucket[cur / sqrtN].update(cur, endOfBucket, val);
                        cur = endOfBucket + 1;
                }
        }
}
ll query(int left, int right) {
        int cur = left;
        ll rt = 0;
        while (cur <= right) {
                if (cur % sqrtN == 0 && cur + sqrtN - 1 <= right)
                        rt += bucket[cur / sqrtN].totalSum, cur += sqrtN;
                else {
                        int endOfBucket = min(right, bucket[cur / sqrtN].r);
```

```
                        rt += bucket[cur / sqrtN].query(cur, endOfBucket);
                        cur = endOfBucket + 1;
                }
        }
        return rt;
}
```

Big Intger

```
#define ll long long
struct BigInt {
        const int BASE = 1000000000;
        vector<int> v;
        BigInt() {}
        BigInt(long long val) { *this = val; }
        int size() const { return v.size(); }
        bool zero() const { return v.empty(); }
        BigInt operator =(const long long& a) {
                v.clear();
                long long val = a;
                while (val) {
                        v.push_back(val % BASE);
                        val /= BASE;
                }
                return *this;
        }
        BigInt operator =(const BigInt& a) {
                v = a.v;
                return *this;
        }
        bool operator <(const BigInt& a)const {
                if (a.size() != size())
                        return size() < a.size();
                for (int i = size() - 1; i >= 0; i--) {
                        if (v[i] != a.v[i]) return v[i] < a.v[i];
                }
                return false;
        }
        bool operator >(const BigInt& a)const {
                return a < *this;
        }
        bool operator == (const BigInt& a)const {
                return (!(*this < a) && !(a < *this));
        }
        BigInt operator +(const BigInt& a) {
                BigInt b = *this; b += a;
                return b;
        }
        BigInt operator +=(const BigInt& a) {
                int idx = 0, carry = 0;
                while (idx < a.size() || carry) {
                        if (idx < a.size())carry += a.v[idx];
                        if (idx == size())v.push_back(0);
                        v[idx] += carry;
                        carry = v[idx] / BASE;
                        v[idx] %= BASE;
                        idx++;
                }
                return *this;}
```

```
	BigInt operator *(const BigInt& a) {
		BigInt res;
		if (this->zero() || a.zero())return res;
		res.v.resize(size() + a.size());
		for (int i = 0; i < size(); i++) {
			if (v[i] == 0)continue;
			ll carry = 0;
			for (int j = 0; carry || j < a.size(); j++) {
				carry += 1LL * v[i] * (j < a.size() ? a.v[j] : 0);
				while (i + j >= res.size())
					res.v.push_back(0);
				carry += res.v[i + j];
				res.v[i + j] = carry % BASE;
				carry /= BASE;
			}
		}
		while (!res.v.empty() && res.v.back() == 0)res.v.pop_back();
		return res;
	}
	friend ostream& operator<<(ostream& stream, const BigInt& a) {
		stream << (a.zero() ? 0 : a.v.back());
		for (int i = (int)a.v.size() - 2; i >= 0; i--)
			stream << setfill('0') << setw(9) << a.v[i];
		return stream;
	}
};
```

Graphs

shortest path algorithms

Dijkstra

```
vector<vector<edge>> adj;
//O(n*log(m))
void dijkstra(int src, int dest = -1) {
	priority_queue<edge> q;
	vector<int> dis(adj.size(), INT_MAX), prev(adj.size(), -1);
	q.push(edge(-1, src, 0));
	dis[src] = 0;
	while (!q.empty()) {
		edge e = q.top(); q.pop();
		if (e.weight > dis[e.to])continue;
		prev[e.to] = e.from;
		for (edge ne : adj[e.to])
			if (dis[ne.to] > dis[e.to] + ne.weight) {
				ne.weight = dis[ne.to] = dis[e.to] + ne.weight;
				q.push(ne);
			}
	}
	vector<int> path;
	while (dest != -1) {
		path.push_back(dest);
		dest = prev[dest];
```

```
	}
	reverse(path.begin(), path.end());
}
```

Bellmanford

```
#define oo 0x3f3f3f3fLL

vector<edge> edgeList;
//O(n*m)
void bellmanford(int n, int src, int dest = -1) {
	vector<int> dis(n + 1, oo), prev(n + 1, -1);
	dis[src] = 0;
	bool negativeCycle = false;
	int last = -1, tmp = n;
	while (tmp--) {
		last = -1;
		for (edge e : edgeList)
			if (dis[e.to] > dis[e.from] + e.weight) {
				dis[e.to] = dis[e.from] + e.weight;
				prev[e.to] = e.from;
				last = e.to;
			}
		if (last == -1) break;
		if (tmp == 0)negativeCycle = true;
	}
	if (last != -1) {
		for (int i = 0; i < n; i++)
			last = prev[last];
		vector<int> cycle;
		for (int cur = last; cur != last || cycle.size() > 1; cur = prev[cur])
			cycle.push_back(cur);
		reverse(cycle.begin(), cycle.end());
	}
	vector<int> path;
	while (dest != -1) {
		path.push_back(dest);
		dest = prev[dest];
	}
	reverse(path.begin(), path.end());
}
```

Difference constraints

```
#include"Bellmanford.h"

void  difference_constraints() {
	int m; cin >> m;
	int cnt = 1;
```

```
	while (m--) {
		string x1, x2; int w; // x1 - x2 <= w
		cin >> x1 >> x2 >> w;
		map<string, int> id;
		if (id.find(x1) == id.end())
			id[x1] = cnt++;
		if (id.find(x2) == id.end())
			id[x2] = cnt++;
		edgeList.emplace_back(id[x2], id[x1], w);
	}
	for (int i = 1; i < cnt; i++) edgeList.emplace_back(cnt, i, 0);
	bellmanford(cnt, cnt);
}
```

Floyd

```
vector<vector<int>> adj, par;
// adj[i][j] = oo , adj[i][i] = 0
// par[i][j] = i
void floyd() {
	for (int k = 1; k < adj.size(); k++)
		for (int i = 1; i < adj.size(); i++)
			for (int j = 1; j < adj.size(); j++)
				if (adj[i][j] > adj[i][k] + adj[k][j]) {
					adj[i][j] = adj[i][k] + adj[k][j];
					par[i][j] = par[k][j];
				}
}
void buildPath(int src, int dest) {
	vector<int> path;
	while (src != dest) {
		path.push_back(dest);
		dest = par[src][dest];
	}
	path.push_back(src);
	reverse(path.begin(), path.end());
}
```

SPFA (shortest path faster algorithm)

```
void SPFA(vector<vector<pair<int, int>>> adjL, int Src, int n, int m) {    //
relative to Bellman_ford
	int Max_Path = INT_MAX;
	vector<int> d(n + 1, Max_Path), cnt(n + 1), prev(n + 1, -1);
	vector<bool> inqueue(n + 1);
	queue <int> q;
	q.push(Src);
	d[Src] = 0;
	inqueue[Src] = 1;
```

```
	int x;
	bool flag = 0;
	while (!q.empty()) {
		int u = q.front();
		q.pop();
		inqueue[u] = 0;
		for (auto it : adjL[u]) {
			int v = it.first;
			int cost = it.second;
			if (d[v] > d[u] + cost) {
				d[v] = max(-Max_Path, d[u] + cost);
				prev[v] = u;
				cnt[v]++;
				if (!inqueue[v]) {
					inqueue[v] = 1;
					q.push(v);
				}
				if (cnt[v] > n) {
					x = v;
					flag = 1;
					break;
				}
			}
		}
		if (flag)
			break;
	}
	if (!flag)
		cout << "No negative cycle from " << Src << endl;
	else {
		int y = x;
		for (int i = 0; i < n; i++)
			y = prev[y];
		vector<int> path;
		for (int cur = prev[y]; ; cur = prev[cur]) {
			path.push_back(cur);
			if (cur == y && path.size() > 1)
				break;
		}
		cout << "Negative cycle: ";
		for (auto it : path)
			cout << it << ' ';
		cout << endl;
	}
}
```

Tarjan

Strongly connected component

```
vector<vector<int>> adj, scc;
vector<set<int>> dag;
vector<int> dfs_num, dfs_low, compId;
vector<bool> inStack;
stack<int> stk;
int timer;

//O(n + m)
void tarjan(int node) {
	dfs_num[node] = dfs_low[node] = ++timer;
	stk.push(node);
	inStack[node] = 1;
	for (int child : adj[node])
		if (!dfs_num[child]) {
			tarjan(child);
			dfs_low[node] = min(dfs_low[node], dfs_low[child]);
		}
		else if (inStack[child])
			dfs_low[node] = min(dfs_low[node], dfs_num[child]);
	//can be dfs_low[node] = min(dfs_low[node], dfs_low[child]);
	if (dfs_low[node] == dfs_num[node]) {
		scc.push_back(vector<int>());
		int v = -1;
		while (v != node) {
			v = stk.top(); stk.pop();
			inStack[v] = 0;
			scc.back().push_back(v);
			compId[v] = scc.size() - 1;
		}
	}
}

void SCC() {
	timer = 0;
	dfs_num = dfs_low = compId = vector<int>(adj.size());
	inStack = vector<bool>(adj.size());
	scc = vector<vector<int>>();
	for (int i = 1; i < adj.size(); i++)
		if (!dfs_num[i]) tarjan(i);
}
void DAG() {
	dag = vector<set<int>>(scc.size());
	for (int i = 1; i < adj.size(); i++)
		for (int j : adj[i]) if (compId[i] != compId[j])
			dag[compId[i]].insert(compId[j]);}
```

Articulation point and bridge

```
vector<vector<int>> adj;
vector<int> dfs_num, dfs_low;
vector<bool> articulation_point;
vector<pair<int, int>> bridge;
int timer, cntChild;

// O(n + m)
void tarjan(int node, int par) {
	dfs_num[node] = dfs_low[node] = ++timer;
	for (int child : adj[node])
		if (!dfs_num[child]) {
			if (par == -1)cntChild++;
			tarjan(child, node);
			if (dfs_low[child] >= dfs_num[node])
				articulation_point[node] = 1;

			if (dfs_low[child] > dfs_num[node])
				bridge.push_back({ node,child });
			dfs_low[node] = min(dfs_low[node], dfs_low[child]);
		}
		else if (child != par)
			dfs_low[node] = min(dfs_low[node], dfs_num[child]);
}

void articulation_point_and_bridge() {
	timer = 0;
	dfs_num = dfs_low = vector<int>(adj.size());
	articulation_point = vector<bool>(adj.size());
	bridge = vector<pair<int, int>>();
	for (int i = 1; i < adj.size(); i++)
		if (!dfs_num[i]) {
			cntChild = 0;
			tarjan(i, -1);
			articulation_point[i] = cntChild > 1;
		}
}
```

Edge Classification

```
vector<vector<int>> adj;
vector<int> start, finish;
int timer;

void dfsEdgeClassification(int node) {
	start[node] = timer++;
	for (int child : adj[node]) {
```

```
            if (start[child] == -1)
                  dfsEdgeClassification(child);
            else {
                  if (finish[child] == -1); // Back Edge
                  else if (start[node] < start[child]); // Forward Edge
                  else; // Cross Edge
            }
      }
      finish[node] = timer++;
}
```

Kruskal (minimum spanning tree)

```
#include"..\data_structures\disjoint_set_union\DSU.h"

vector<edge> edgeList;
//O(m*log(m))
pair<int, vector<edge>> MST_Kruskal(int n) {
      DSU uf(n);
      vector<edge> edges;
      int mstCost = 0;
      sort(edgeList.begin(), edgeList.end());
      for (auto e : edgeList)
            if (uf.union_sets(e.from, e.to)) {
                  mstCost += e.weight;
                  edges.push_back(e);
            }
      if (edges.size() != n - 1)return { INT_MAX,vector<edge>() };
      return { mstCost,edges };
}

int miniMax(int src, int dest, int n) {
      int max = INT_MIN;
      DSU uf(n);
      sort(edgeList.begin(), edgeList.end());
      for (auto e : edgeList) {
            if (uf.same_set(src, dest))return max;
            uf.union_sets(e.from, e.to);
            max = e.weight;
      }
      return max;
}
```

2 SAT

```
#include"tarjan/strongly_connected_component.h"

int n;
int Not(int x) {
      return(x > n ? x - n : x + n);
}
```

```
void addEdge(int a, int b) {
	adj[Not(a)].push_back(b);
	adj[Not(b)].push_back(a);
}

bool _2SAT(vector<int>& value) {
	SCC();
	for (int i = 1; i <= n; i++)
		if (compId[i] == compId[Not(i)])
			return false;
	vector<int> assign(scc.size(), -1);
	for (int i = 0; i < scc.size(); i++) if (assign[i] == -1) {
		assign[i] = true;
		assign[compId[Not(scc[i].back())]] = false;
	}
	for (int i = 1; i <= n; i++)value[i] = assign[compId[i]];
	return true;
}
```

Maximum bipartite matching

```
vector<vector<bool>> adjMat;
vector<vector<int>> adj;
vector<int> rowAssign, colAssign, vis;
int test_id;
bool canMatch(int i) {
	for (int j : adj[i]) if (vis[j] != test_id) {
		vis[j] = test_id;
		if (colAssign[j] == -1 || canMatch(colAssign[j])) {
			colAssign[j] = i; rowAssign[i] = j;
			return true;
		}
	}
	return false;
}

// O(rows * E)
int maximum_bipartite_matching(int rows, int cols) {
	int maxFlow = 0;
	rowAssign = vector<int>(rows, -1);
	colAssign = vector<int>(cols, -1);
	vis = vector<int>(cols);
	for (int i = 1; i < rows; i++) {
		test_id++;
		if (canMatch(i))maxFlow++;
	}
	vector<pair<int, int>> matches;
	for (int j = 1; j < cols; j++) if (~colAssign[j])
		matches.push_back({ colAssign[j],j });
	return maxFlow;
}
```

Geometry

Points

```
/*
```

```
// Polar system , Cartesian
x = r * cos(O)
y = r * sin(O)
r = sqrt(x^2  + y^2)
O = atan2(y,x)
// Rotatet
x_  = cos(O) -sin(O) * x
y_  = sin(O) - cos(O) * y
//vectors
Vector = Direction + Magnitude
Two vectors are perpendicular if and only if their angle is a right angle
Set of vectors is orthogonal if and only if they are pairwise perpendicular
The normal vector to a surface is a vector which is perpendicular to the surface at a
given point

Dot Product : Algebraically, sum of the products of the corresponding entries
Geometrically, the product of the Euclidean magnitudes of the two vectors
and the cosine of the angle between them.
A . B = |A| |B| cos(O) = x1*x2 + y1*y2
if A and B are orthogonal, then the angle between them is 90  A.B = 0
if they are codirectional, then the angle between them is 0   A.B = |A| |B|
if (O)  > 90 then A.B <0  and if(O) < 90 then A.B > 0 if (O) = 90 the A.B = 0

The cross product, a X b, is a vector that is perpendicular
to both a and b and therefore normal to the plane containing them.
-one if the two are perpendicular and a magnitude of zero if the two are parallel.
A x B  = A.x * B.y - B.x * A.y = r1 * r2 * sin(T2 - T1)

//complex numbers
point a(2,3)  >> norm(a)  = 2^2 + 3^2 = 13
conj(a)  >> 2 + 3i  > 2 -3i  flip sign i
*/
typedef complex<double> point;  // it can be long long not double
template<class T>
istream& operator>> (istream& is, complex<T>& p) {
      T value;
      is >> value;
      p.real(value);
      is >> value;
      p.imag(value);
      return is;
}
#define PI acos(-1.0)
#define EPS 1e-8
#define X real()   // can sign values point a; a.real(5) , a.image(2)
#define Y imag()
#define angle(a)  (atan2((a).imag(), (a).real())) // angle with orignial
```

```
#define dist(a)   (hypot((a).imag(), (a).real())) // distance between two point send
diff
#define length(a) dist(a)
#define vec(a,b)  ((b)-(a)) // diff x1-x2 , y1-y2 return vec (x,y)
#define rotateO(p,ang)    ((p)*exp(point(0,ang)))  // angle should be in radian aroun
origin
#define rotateA(p,ang,about)  (rotateO(vec(about,p),ang)+about)// rotate around point
#define same(p1,p2)             (dp(vec(p1,p2),vec(p1,p2)) < EPS) // check to points
same or not
#define dp(a,b)   ( (conj(a)*(b)).real() )    // a*b cos(T), if zero -> prep dot
product A.B
#define cp(a,b)   ( (conj(a)*(b)).imag() )    // a*b sin(T), if zero -> parllel cross
product = area of parllelogram
#define norm(a)  (norm(a))  // return x^2 + y^2 a is point can use dp(a,a)
#define reflectO(v,m)  (conj((v)/(m))*(m))
#define normalize(a)            (a)/dist(a)
double toRadians(double degree) {
	return (degree * PI) / 180.0;
}
int dcmp(long double x, long double y) {
	return fabs(x - y) <= EPS ? 0 : x < y ? -1 : 1;
}
double fixAngle(double A) {
	return A > 1 ? 1 : (A < -1 ? -1 : A);
}
double fixMod(double a, double b) {
	return fmod(fmod(a, b) + b, b);
}
point translate(point p, point v) { // translate p according to v
	return point(p.X + v.X, p.Y + v.Y);
}
point scale(point v, double s) { // nonnegative s = [<1 .. 1 .. >1]
	return point(v.X * s, v.Y * s);
} // shorter.same.longer
// when sort points
bool cmp(point a, point b) {
	if (fabs(a.X - b.X) < EPS) {
		return a.Y < b.Y;
	}
	return a.X < b.X;
}
point reflect(point p, point p0, point p1) {
	point z = p - p0, w = p1 - p0;
	return conj(z / w) * w + p0;  // Refelect point p1 around p0p1
}
// return min angle: aOb / bOa
// dp(v1, v2) = |v1|*|v2|*cos(theta)
double angleO(point a, point O, point b) {
```

```
	point v1(a - O), v2(b - O);
	return acos(fixAngle(dp(v1, v2) / dist(v1) / dist(v2)));
}
double getAng(point& a, point& b, point& c) // find angle abc, anticlock bc to ba
{
	double ang = angle(vec(b, c)) - angle(vec(b, a));
	//if (dcmp(ang, 0) < 0)
	ang += 2 * PI;
	return ang;
}
```

Lines

```
#include"points.h"
/*
equation
explicit 2D  y = mx + b  , m = (y2-y1)/(x2-x1) ,get b from given point1, or point2
Implicit 2D  ax + by + c = 0  . a = y1 - y2 , b = x2 - x1 , c = x1y2 - x2y1
Parametric  P(t) = P0 + tVL
collinear if slop1 = slop2
perpendicular  if slop1 * slop2 = -1

if point is over line or not
y = mx + c  , point (x0,y0) get m , c
y0 - mx0+c > 0 then above
y0 - mx0+c < 0 the below
y0 - mx0+c = 0  then over

intersection of two lines
y1= mx1 + c1 , y2 = mx2 + c2
mx1+c1 = mx2 + c2  get x2 then get y1 from any equation

*/
struct line {
	double a, b, c;
};
void pointsToLine(point p1, point p2, line& l) {
	if (fabs(p1.X - p2.X) < EPS) { // vertical line is fine
		l.a = 1.0; l.b = 0.0; l.c = -p1.X; // default values
	}
	else {
		l.a = -(double)(p1.Y - p2.Y) / (p1.X - p2.X);
		l.b = 1.0; // IMPORTANT: we fix the value of b to 1.0
		l.c = -(double)(l.a * p1.X) - p1.Y;
	}
}
bool areParallel(line l1, line l2) { // check coefficients a & b
	return (fabs(l1.a - l2.a) < EPS) && (fabs(l1.b - l2.b) < EPS);
```

```
}
bool areSame(line l1, line l2) { // also check coefficient c
	return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS);
}
bool areIntersect(line l1, line l2, point& p) {
	if (areParallel(l1, l2)) return false; // no intersection
	// solve system of 2 linear algebraic equations with 2 unknowns
	p.real((l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b));
	// special case: test for vertical line to avoid division by zero
	if (fabs(l1.b) > EPS) p.imag(-(l1.a * p.X + l1.c));
	else p.imag(-(l2.a * p.X + l2.c));
	return true;
}
bool isCollinear(point a, point b, point c) {
	return fabs(cp(b - a, c - a)) < EPS;
}
// point c inRay a-b->
bool isPointOnRay(point a, point b, point c) {
	if (dist(vec(a, c)) < EPS) return true;
	return same(normalize(vec(a, b)), normalize(vec(a, c)));
}
// point c inSegment a-b
bool isPointOnSegment(point a, point b, point c) {
	double acb = length(vec(b, a)), ac = length(vec(c, a)), cb = length(vec(c,
b));
	return dcmp(acb - (ac + cb), 0) == 0;
}
// dist point p2 to line p0-p1
double distToLine(point p0, point p1, point p2) {
	return fabs(cp(p1 - p0, p2 - p0) / dist(p0 - p1)); // area = 0.5*b*h
}
// distance from point p2 to segment p0-p1
// p4 is the nearest point to p2
double distToSegment(point p0, point p1, point p2, point& p4) {
	double d1, d2;
	point v1 = p1 - p0, v2 = p2 - p0;
	if ((d1 = dp(v1, v2)) <= 0) {
		p4 = p0;
		return dist(p2 - p0);
	}
	if ((d2 = dp(v1, v1)) <= d1) {
		p4 = p1;
		return dist(p2 - p1);
	}
	double t = d1 / d2;
	p4 = (p0 + v1 * t);  // this is point
	return dist(p2 - (p0 + v1 * t));
}
```

```
bool intersectSegments(point a, point b, point c, point d, point& intersect) {
	double d1 = cp(a - b, d - c), d2 = cp(a - c, d - c), d3 = cp(a - b, a - c);
	if (fabs(d1) < EPS)
		return false;  // Parllel || identical

	double t1 = (double)d2 / d1, t2 = d3 / d1;
	intersect = a + (b - a) * t1;

	if (t1 > 1 + EPS || t1 < -EPS || t2 < -EPS || t2 > 1 + EPS)
		return false;  //e.g ab is ray, cd is segment ... change to whatever
	return true;
}
// Where is c relative to segment a-b?
// ccw = +1 => angle > 0 or collinear after b
// point c is counter-clockwise about segment a-b
// cw = -1 => angle < 0 or collinear after a
// point c is clockwise about segment a-b
// Undefined = 0 => Collinar in range [a, b]. Be careful here
int ccw(point a, point b, point c) {
	point v1(b - a), v2(c - a);
	double t = cp(v1, v2);

	if (t > +EPS)
		return 1;
	if (t < -EPS)
		return -1;
	if (v1.X * v2.X < -EPS || v1.Y * v2.Y < -EPS)
		return -1;
	if (norm(v1) < norm(v2) - EPS)
		return +1;
	return 0;
}
bool intersect(point p1, point p2, point p3, point p4) {
	// special case handling if a segment is just a point
	bool x = (p1 == p2), y = (p3 == p4);
	if (x && y)  return p1 == p3;
	if (x)   return ccw(p3, p4, p1) == 0;
	if (y)   return ccw(p1, p2, p3) == 0;

	return  ccw(p1, p2, p3) * ccw(p1, p2, p4) <= 0 &&
		ccw(p3, p4, p1) * ccw(p3, p4, p2) <= 0;
}

bool lineInsideRectangle(double x1, double x2, double y1, double y2, point st, point
ed) {
	if (x2 < x1) swap(x1, x2);
	if (y2 < y1) swap(y1, y2);
	double mnX = min(st.X, ed.X), mxX = max(st.X, ed.X),
```

```
            mnY = min(st.Y, ed.Y), mxY = (st.Y, ed.Y);
      return dcmp(x1, mnX) <= 0 && dcmp(x2, mxX) >= 0 && dcmp(y1, mnY) <= 0 &&
dcmp(y2, mxY) >= 0;
}
```

Triangles

```
#include"points.h"
#include"lines.h"
/*

A triangle with three sides: a, b, c has perimeter p = a + b + c and semi-perimeter
s = 0.5 × p

A triangle with 3 sides: a, b, c and semi-perimeter s has
area A = sqrt(s × (s − a) × (s − b) × (s − c));

A triangle with area A and semi-perimeter s has an inscribed circle (incircle) with
radius r = A/s

Law of Sines
a/sin(α) = b/ sin(b) = c/sin(c) = 2R
c^2= a^2 + b^2 − 2 × a × b × cos(γ)

A trapezium with a pair of parallel edges of lengths w1 and w2; and a height h
between
both parallel edges has area A = 0.5 × (w1 + w2) × h
*/
// sin(A)/a = sin(B)/b = sin(C)/c
// a^2 = b^2 + c^2 - 2b*c*cos(A)

double getSide_a_bAB(double b, double A, double B) {
      return (sin(A) * b) / sin(B);
}

double getAngle_A_abB(double a, double b, double B) {
      return asin(fixAngle((a * sin(b)) / b));
}

// gave me WR answer in team formation :D
double getAngle_A_abc(double a, double b, double c) {
      return acos(fixAngle((b * b + c * c - a * a) / (2 * b * c)));
}
double perimeter_triangle(double a, double b, double c) {
      return a + b + c;
}
double area_triangle(double a, double b, double c) {
      double s = 0.5 * perimeter_triangle(a, b, c);
```

```
	return sqrt(s * (s - a) * (s - b) * (s - c));
}
double triangleArea(point p0, point p1, point p2) {
	double a = length(vec(p1, p0)), b = length(vec(p2, p0)),
		c = length(vec(p2, p1));
	return triangleArea(a, b, c);
}
double rInCircle(double ab, double bc, double ca) {
	return area_triangle(ab, bc, ca) / (0.5 * perimeter_triangle(ab, bc, ca));
}
double rInCircle(point a, point b, point c) {
	return rInCircle(dist(a - b), dist(b - c), dist(c - a));
}
// Get radius and point of circle that inscribed with triangle
// returns 1 if there is an inCircle center, returns 0 otherwise
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point& ctr, double& r) {
	r = rInCircle(p1, p2, p3);
	if (fabs(r) < EPS) return 0; // no inCircle center
	line l1, l2; // compute these two angle bisectors
	double ratio = dist(p1 - p2) / dist(p1 - p3);
	point p = translate(p2, scale(vec(p2, p3), ratio / (1 + ratio)));
	pointsToLine(p1, p, l1);
	ratio = dist(p2 - p1) / dist(p2 - p3);
	p = translate(p1, scale(vec(p1, p3), ratio / (1 + ratio)));
	pointsToLine(p2, p, l2);
	areIntersect(l1, l2, ctr); // get their intersection point
	return 1;
}

double rCircumCircle(double ab, double bc, double ca) {
	return ab * bc * ca / (4.0 * area_triangle(ab, bc, ca));
}
double rCircumCircle(point a, point b, point c) {
	return rCircumCircle(dist(a - b), dist(b - c), dist(c - a));
}

double polygon_area(vector<point>points) {
	double area = 0;
	for (int i = 0; i < points.size() - 1; i++) {
		area += cp(vec(points[0], points[i]), vec(points[0], points[i + 1]));
		//area += area_triangle(dist(points[0] - points[i]), dist(points[0] -
points[i+1]),
		//	dist(points[i] - points[i+1]));
		//point p1 = i ? points[i - 1] : points.back(),p2 = points[i];
		//area += (p1.X - p2.X) * (p1.Y + p2.Y);
	}
```

```
	return abs(area / 2.0);
}
```

Circles

```
#include"points.h"
#include"lines.h" //intersectSegments
#include"triangles.h" //getAngle_A_abc
/*
formala  (x-h) ^ 2 + (y-k)^2 = r^2
(h,k) is center, (x,y) any point in circle

// If line intersect cirlce at point p, and p = p0 + t(p1-p0)
// Then (p-c)(p-c) = r^2 substitute p and rearrange
// (p1-p0)(p1-p0)t^2 + 2(p1-p0)(p0-C)t + (p0-C)(p0-C) = r*r; -> Quadratic
*/

//(x-h) ^ 2 + (y-k)^2 = r^2
bool is_insideCircle(point center, point b, double r) {
	double d1 = (b.X - center.X);
	double d2 = (b.Y - center.Y);
	return (d1 * d1 + d2 * d2) <= r * r;
}
bool circle2PtsRad(point p1, point p2, double r, point& c) {
	double d2 = (p1.X - p2.X) * (p1.X - p2.X) +
		(p1.Y - p2.Y) * (p1.Y - p2.Y);
	double det = r * r / d2 - 0.25;
	if (det < 0.0) return false;
	double h = sqrt(det);
	c.real((p1.X + p2.X) * 0.5 + (p1.Y - p2.Y) * h);
	c.imag((p1.Y + p2.Y) * 0.5 + (p2.X - p1.X) * h);
	return true;
	// to get the other center, reverse p1 and p2
}
// 2 points has infinite circles
// Find circle passes with 3 points, some times, there is no circle! (in case
colinear)
// Draw two perpendicular lines and intersect them
pair<double, point> findCircle(point a, point b, point c) {
	//create median, vector, its prependicular
	point m1 = (b + a) * 0.5, v1 = b - a, pv1 = point(v1.Y, -v1.X);
	point m2 = (b + c) * 0.5, v2 = b - c, pv2 = point(v2.Y, -v2.X);
	point end1 = m1 + pv1, end2 = m2 + pv2, center;
	intersectSegments(m1, end1, m2, end2, center);
	return make_pair(length(vec(center, a)), center);
}

// If line intersect cirlce at point p, and p = p0 + t(p1-p0)
```

```
// Then (p-c)(p-c) = r^2 substitute p and rearrange
// (p1-p0)(p1-p0)t^2 + 2(p1-p0)(p0-C)t + (p0-C)(p0-C) = r*r; -> Quadratic
vector<point> intersectLineCircle(point p0, point p1, point C, double r) {
	double a = dp(vec(p0, p1), vec(p0, p1)), b = 2 * dp(vec(p0, p1), vec(C, p0)),
		c = dp(vec(C, p0), vec(C, p0)) - r * r;
	double f = b * b - 4 * a * c;
	vector<point> v;
	if (dcmp(f, 0) >= 0) {
		if (dcmp(f, 0) == 0)	f = 0;
		double t1 = (-b + sqrt(f)) / (2 * a);
		double t2 = (-b - sqrt(f)) / (2 * a);
		v.push_back(p0 + t1 * (p1 - p0));
		if (dcmp(f, 0) != 0)	v.push_back(p0 + t2 * (p1 - p0));
	}
	return v;
}

vector<point> intersectCircleCircle(point c1, double r1, point c2, double r2) {
	// Handle infinity case first: same center/radius and r > 0
	if (same(c1, c2) && dcmp(r1, r2) == 0 && dcmp(r1, 0) > 0)
		return vector<point>(3, c1);	// infinity 2 same circles (not points)

	  // Compute 2 intersection case and handle 0, 1, 2 cases
	double ang1 = angle(vec(c1, c2)), ang2 = getAngle_A_abc(r2, r1, length(vec(c1,
c2)));

	if (::isnan(ang2)) // if r1 or d = 0 => nan in getAngle_A_abc (/0)
		ang2 = 0; // fix corruption

	vector<point> v(1, polar(r1, ang1 + ang2) + c1);

	// if point NOT on the 2 circles = no intersection
	if (dcmp(dp(vec(c1, v[0]), vec(c1, v[0])), r1 * r1) != 0 ||
		dcmp(dp(vec(c2, v[0]), vec(c2, v[0])), r2 * r2) != 0)
		return vector<point>();

	v.push_back(polar(r1, ang1 - ang2) + c1);
	if (same(v[0], v[1]))  // if same, then 1 intersection only
		v.pop_back();
	return v;
}

bool is_intersect_circles(double x1, double y1, double r1, double x2, double y2,
double r2) {
	double x = x1 - x2;
	double y = y1 - y2;
	double dist = sqrt(x * x + y * y);
	return dist < (r1 + r2) && (abs(r1 - r2) <= dist);
```

```
}
double distance(double x1, double y1, double x2, double y2) {
	double xx = (x1 - x2);
	double yy = (y1 - y2);
	return (xx * xx) + (yy * yy);
}
//get center point of line  with radious
pair<double, double> center(double x1, double y1, double x2, double y2, double rr) {
	double ab = distance(x1, y1, x2, y2);
	double k = sqrt(rr / ab - 0.25);
	pair<double, double> o;
	o.first = (x1 + x2) / 2.0 + k * (y2 - y1);
	o.second = (y1 + y2) / 2.0 + k * (x1 - x2);
	return o;
}
```

Math's

Elementary

```
#define ll long long
#define EPS 1e-8
#define numOfDigit(x) 1+(int)(floor(log10(x)))
#define numOfBits(x) 1+(int)(floor(log2(x)))

int dcmp(double x, double y) { return fabs(x - y) <= EPS ? 0 : x < y ? -1 : 1; }
ll gcd(ll a, ll b) { return !b ? abs(a) : gcd(b, a % b); }
ll lcm(ll a, ll b) { return abs(a / gcd(a, b)) * b; }
//return sum of sequence a, a+x , a+2x .... b
ll sequence(ll a, ll b, ll x) {
	a = ((a + x - 1) / x) * x;
	b = (b / x) * x;
	return (b + a) * (b - a + x) / (2 * x);
}

ll power(ll x, ll y) {
	if (y == 0) return 1;
	if (y == 1) return x;
	ll r = power(x, y >> 1);
	return r * r * power(x, y & 1);
}

//sum 1/(x^i) for i = 1 to n
double summation(int x, int n) {
	double p = power(x, n);
	return(p - (x - 1.0)) / p;
}
```

Primes

```
#define ll long long

// check number is prime or not
// O(sqrt(n))
bool isprime(ll num) {
	if (num == 2) return true;
	if (num < 2 || !(num & 1)) return false;
	for (ll i = 3; i * i <= num; i += 2)
		if (num % i == 0) return false;
	return true;
}

const int N = 1e8;
bool isPrime[N + 1];
vector<int> prime;
// check all numbers from 1 to n prime or not
// O(n*log(log(n)))
void sieve() {
	memset(isPrime, true, sizeof(isPrime));
	isPrime[0] = isPrime[1] = false;
	for (int i = 4; i <= N; i += 2) isPrime[i] = false;
	for (int i = 3; i * i <= N; i += 2) {
		if (isPrime[i])
			for (int j = i * i; j <= N; j += i + i)
				isPrime[j] = false;
	}
	prime.push_back(2);
	for (int i = 3; i <= N; i += 2)
		if (isPrime[i]) prime.push_back(i);
}


// generate prime divisors for all number from 1 to n
// O(n*log(n)) // max -> 2e6
const int M = 2e6;
vector<int> primeDivs[M + 1];
void primeDivisors() {
	for (int i = 2; i <= M; i += 2)
		primeDivs[i].push_back(2);
	for (int i = 3; i <= M; i += 2) {
		if (primeDivs[i].empty())
			for (int j = i; j <= M; j += i)
				primeDivs[j].push_back(i);
	}
}
```

Prime Factorization

```
#include"elementary.h" //power
#include"primes.h" //sieve
using namespace std;
#define ll long long
typedef vector<pair<ll, int>> primeFactors;

// generate prime divisors in n
// n = p1^x1 * p2^x2 .... pn^xn
// O(sqrt(n)) // max = 1e16
primeFactors prime_factors(ll n) {
	primeFactors p;
	int idx = 0;
	while (!(n <= N && isPrime[n]) && idx < prime.size() && (ll)prime[idx] *
prime[idx] <= n) {
		int cnt = 0;
		while (n % prime[idx] == 0)
			n /= prime[idx], cnt++;
		if (cnt) p.push_back({ prime[idx],cnt });
		idx++;
	}
	if (n > 1)p.push_back({ n,1 });
	return p;
}

//return multiplication of tow nember using prime factorization
primeFactors multiplication(primeFactors& a, primeFactors& b) {
	primeFactors rt;
	int i = 0, j = 0;
	while (i < a.size() && j < b.size()) {
		if (a[i].first < b[j].first) {
			rt.emplace_back(a[i]);i++;
		}
		else if (a[i].first > b[j].first) {
			rt.emplace_back(b[j]);j++;
		}
		else {
			rt.emplace_back(a[i].first, a[i].second + b[j].second);
			i++; j++;
		}
	}
	while (i < a.size()) { rt.push_back(a[i]); i++; }
	while (j < b.size()) { rt.push_back(b[j]); j++; }
	return rt;
}

// return gcd between two number using prime factorization
```

```
primeFactors gcd(primeFactors a, primeFactors b) {
	primeFactors gcd;
	int i = 0, j = 0;
	while (i < a.size() && j < b.size()) {
		if (a[i].first < b[j].first)i++;
		else if (a[i].first > b[j].first)j++;
		else {
			gcd.push_back({ a[i].first,min(a[i].second,b[j].second) });
			i++; j++;
		}
	}
	return gcd;
}
// return lcm between two number using prime factorization
primeFactors lcm(primeFactors a, primeFactors b) {
	primeFactors lcm;
	int i = 0, j = 0;
	while (i < a.size() && j < b.size()) {
		if (a[i].first < b[j].first) {
			lcm.push_back(a[i]); i++;
		}
		else if (a[i].first > b[j].first) {
			lcm.push_back(b[j]); j++;
		}
		else {
			lcm.push_back({ a[i].first, max(a[i].second, b[j].second) });
			i++; j++;
		}
	}
	while (i < a.size()) { lcm.push_back(a[i]); i++; }
	while (j < b.size()) { lcm.push_back(b[j]); j++; }
}

// return number of Divisors(n) using prime factorization
ll numOfDivisors(primeFactors mp) {
	ll cnt = 1;
	for (auto it : mp) cnt *= (it.second + 1);
	return cnt;}

// return sum of Divisors(n) using prime factorization
ll sumOfDivisors(primeFactors mp) {
	ll sum = 1;
	for (auto it : mp)
		sum *= (power(it.first, it.second + 1) - 1) / (it.first - 1);
	return sum;
}
```

Factorization

```
#define ll long long

const int N = 1e6;
vector<int> divisors[N + 1];
// generate divisors for all number from 1 to n
// O(n*log(n)) // max-> 1e6
void rangeDivisors() {
	for (int i = 1; i <= N; i++)
		for (int j = i; j <= N; j += i)
			divisors[j].push_back(i);
}

// return sum of divisors for all number from 1 to n
//O(n) // max -> 1e8
ll sumRangeDivisors(int n) {
	ll ans = 0;
	for (int x = 1; x <= n; x++)
		ans += (n / x) * x;
	return ans;
}

// return sum of divisors for all number from 1 to n
// max -> 1e9
ll get_sum_div(ll x) {
	ll ans = 0, left = 1, right;
	for (; left <= x; left = right + 1) {
		right = x / (x / left);
		ans += (x / left) * (left + right) * (right - left + 1) / 2;
	}
	return ans;
}
```

Mod Inverse

```
#define ll long long

ll power(ll x, ll y, int mod) {
	if (y == 0) return 1;
	if (y == 1) return x % mod;
	ll r = power(x, y >> 1, mod);
	return (((r * r) % mod) * power(x, y & 1, mod)) % mod;
}

// (a / b) % mod = (a% mod) * (b ^ (mod - 2))%mod
// Modular inverse of the given number modulo mod
// return z = (1/b) % mod // mod must be Prime
ll modInverse(ll b, ll mod) {
	return power(b, mod - 2, mod);
}
```

```
// Calulate Modular inverse
ll modInv(ll a, ll m) {
	ll m0 = m, t, q;
	ll x0 = 0, x1 = 1;
	if (m == 1)
		return 0;
	while (a > 1) {
		q = a / m;
		t = m;
		m = a % m, a = t;
		t = x0;
		x0 = x1 - q * x0;
		x1 = t;
	}
	if (x1 < 0)
		x1 += m0;
	return x1;
}

const int N = 1e5 + 100;
const int mod = 1e9 + 7;
ll fact[N];
ll inv[N];//mod inverse for i
ll invfact[N];//mod inverse for i!
void factInverse() {
	fact[0] = inv[1] = fact[1] = invfact[0] = invfact[1] = 1;
	for (long long i = 2; i < N; i++) {
		fact[i] = (fact[i - 1] * i) % mod;
		inv[i] = mod - (inv[mod % i] * (mod / i) % mod);
		invfact[i] = (inv[i] * invfact[i - 1] % mod);
	}
}
```

Combinatorics

```
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;

/*
nCr = n!/((n-r)! * r!)
nPr = n!/(n-r)!
nPr(circle) = nPr/r
*/

ull nCr(int n, int r) {
	if (r > n)return 0;
	ull ans = 1, div = 1, i = r + 1;
	while (i <= n) { ans *= i++; ans /= div++; }
	return ans;
}

ull nPr(int n, int r) {
	if (r > n)return 0;
	ull p = 1, i = n - r + 1;
	while (i <= n) p *= i++;
	return p;
```

```
}

// return nCr using pascal triangle
vector<vector<ull>> Pascal;
ull pascalTriangle(int n, int r) {
	if (r > n || n < 0 || r < 0)return 0;
	ull& rt = Pascal[n][r];
	if (rt)return rt;
	if (r == 0 || n == r) return rt = 1;
	rt = pascalTriangle(n - 1, r) + pascalTriangle(n - 1, r - 1);
	return rt;
}


// return catalan number n-th using dp O(n^2)
// catalan[n] = nCr(2n,n)/(n+1) //max = 35
vector<ull> catalan;
ull catalanNumber(int n) {
	if (n <= 1)return 1;
	ull& rt = catalan[n];
	if (rt)return rt;
	for (int i = 0; i < n; i++)
		rt += catalanNumber(i) * catalanNumber(n - i - 1);
	return rt;
}

// count number of paths in matrix n*m
// go to right or down only
ull countNumberOfPaths(int n, int m) {
	return nCr(n + m - 2, n - 1);
}
```

Matrices

```
#define ll long long
#define sz(v) (int)(v.size())

typedef vector<int> row;
typedef vector<row> matrix;

matrix initial(int n, int m, int val = 0) {
	return matrix(n, row(m, val));
}

matrix identity(int n) {
	matrix rt = initial(n, n);
	for (int i = 0; i < n; i++)rt[i][i] = 1;
	return rt;
}

matrix addIdentity(const matrix& a) {
	matrix rt = a;
	for (int i = 0; i < sz(a); i++)rt[i][i] += 1;
	return rt;
```

```
}

matrix add(const matrix& a, const matrix& b) {
	matrix rt = initial(sz(a), sz(a[0]));
	for (int i = 0; i < sz(a); i++)for (int j = 0; j < sz(a[0]); j++)
		rt[i][j] = a[i][j] + b[i][j];
	return rt;
}

matrix multiply(const matrix& a, const matrix& b) {
	matrix rt = initial(sz(a), sz(b[0]));
	for (int i = 0; i < sz(a); i++) for (int k = 0; k < sz(a[0]); k++) {
		if (a[i][k] == 0)continue;
		for (int j = 0; j < sz(b[0]); j++)
			rt[i][j] += a[i][k] * b[k][j];
	}
	return rt;
}

matrix power(const matrix& a, ll k) {
	if (k == 0)return identity(sz(a));
	if (k & 1)return multiply(a, power(a, k - 1));
	return power(multiply(a, a), k >> 1);
}

matrix power_itr(matrix a, ll k) {
	matrix rt = identity(sz(a));
	while (k) {
		if (k & 1)rt = multiply(rt, a);
		a = multiply(a, a); k >>= 1;
	}
	return rt;
}

matrix sumPower(const matrix& a, ll k) {
	if (k == 0)return initial(sz(a), sz(a));
	if (k & 1)return multiply(a, addIdentity(sumPower(a, k - 1)));
	return multiply(sumPower(a, k >> 1), addIdentity(power(a, k >> 1)));
}

matrix sumPowerV2(const matrix& a, ll k) {
	int n = sz(a);
	matrix rt = initial(2 * n, 2 * n);
	for (int i = 0; i < 2 * n; i++)
		for (int j = 0; j < n; j++)
			rt[i][j] = a[i % n][j];
	for (int i = n; i < 2 * n; i++)rt[i][i] = 1;
	return power(rt, k);
```

```
}

ll fibonacciMatrix(ll n) {
	if (n <= 1)return n;
	/*
	transition matrix
	0 1
	1 1

	fibonacci matrix
	0 1
	0 0
	*/
	matrix transition = initial(2, 2);
	transition[0][1] = transition[1][0] = transition[1][1] = 1;
	matrix transtion_n = power(transition, n - 1);
	matrix fibonacci = initial(2, 2);
	fibonacci[0][1] = 1;
	fibonacci = multiply(fibonacci, transtion_n);
	return fibonacci[0][1];
}
```

string processing

KMP

```
vector<int> failure_function(string pattern) {
	int m = pattern.size();
	vector<int> longestPrefix(m);
	for (int i = 1, k = 0; i < m; i++) {
		while (k > 0 && pattern[k] != pattern[i])
			k = longestPrefix[k - 1];
		if (pattern[k] == pattern[i])k++;
		longestPrefix[i] = k;
	}
	return longestPrefix;
}

void KMP(string str, string pattern) {
	int n = str.size();
	int m = pattern.size();
	vector<int> longestPrefix = failure_function(pattern);
	for (int i = 0, k = 0; i < n; i++) {
		while (k > 0 && pattern[k] != str[i])
			k = longestPrefix[k - 1];
		if (pattern[k] == str[i])k++;
		if (k == m) {
			cout << i - m + 1 << endl;
			k = longestPrefix[k - 1]; // if you want next match
```

```
            }
        }
}
```

Trie

```
vector<vector<int>> trie;
vector<bool> leaf;
void addNode() {
        trie.push_back(vector<int>(26, -1));
        leaf.push_back(false);
}

void insert(const string& s) {
        int root = 0;
        for (const char& ch : s) {
                if (trie[root][ch - 'a'] == -1) {
                        trie[root][ch - 'a'] = trie.size();
                        addNode();
                }
                root = trie[root][ch - 'a'];
        }
        leaf[root] = true;
}

bool find(const string& s) {
        int root = 0;
        for (const char& ch : s) {
                if (trie[root][ch - 'a'] == -1)
                        return false;
                root = trie[root][ch - 'a'];
        }
        return leaf[root];
}

struct trie {
        map<char, trie*> nxt;
        bool isLeaf;
        trie() { isLeaf = 0; }
        void insert(char* str) {
                if (*str == '\0') { isLeaf = true; return; }
                char cur = *str;
                if (nxt.find(cur) == nxt.end())
                        nxt[cur] = new trie();
                nxt[cur]->insert(++str);
        }
        bool find(char* str) {
                if (*str == '\0') { return isLeaf; }
                char cur = *str;
                if (nxt.find(cur) == nxt.end())
                        return false;
                return nxt[cur]->find(++str);
        }
        bool prefixExist(char* str) {
                if (*str == '\0') { return true; }
```

```
            char cur = *str;
            if (nxt.find(cur) == nxt.end())
                  return false;
            return nxt[cur]->prefixExist(++str);
      }
};
```

Suffix array

```
#define all(v) v.begin(),v.end()

int n;
vector<int> suf, order, tmp;
int getOrder(int a) {
      return (a < order.size() ? order[a] : 0);
}
void radix_sort(int k) {
      vector<int> frq(n);
      for (auto& it : suf) frq[getOrder(it + k)]++;
      for (int i = 1; i < n; i++)
            frq[i] += frq[i - 1];
      for (int i = n - 1; i >= 0; i--)
            tmp[--frq[getOrder(suf[i] + k)]] = suf[i];
      suf = tmp;
}
struct comp {
      int len;
      comp(int len) :len(len) {}
      bool operator ()(const int& a, const int& b) const {
            if (order[a] != order[b])
                  return order[a] < order[b];
            return getOrder(a + len) < getOrder(b + len);
      }
};
// n*log(n)
void suffixArray(string s) {
      n = s.size() + 1;
      vector<int> newOrder(n);
      for (int i = 0; i < n; i++) {
            suf.push_back(i);
            tmp.push_back(s[i]);
      }
      sort(all(tmp));
      for (int i = 0; i < n; i++)
            order.push_back(lower_bound(all(tmp), s[i]) - tmp.begin());
      for (int len = 1; newOrder.back() != n - 1; len <<= 1) {
            //sort(all(suf), comp(len));
            radix_sort(len);
            radix_sort(0);
            for (int i = 1; i < n; i++)
                  newOrder[i] = newOrder[i - 1] + comp(len)(suf[i - 1], suf[i]);
            for (int i = 0; i < n; i++)
                  order[suf[i]] = newOrder[i];
      }
```

```
}
// return longest Common prefix in suffix array between (i,i-1)
// O(n)
vector<int> LCP(string s) {
	suffixArray(s);
	vector<int> rank(n), lcp(n);
	for (int i = 0; i < n; i++)
		rank[suf[i]] = i;
	int c = 0;
	for (int i = 0; i < n; i++) {
		if (rank[i]) {
			int j = suf[rank[i] - 1];
			while (s[i + c] == s[j + c])c++;
		}
		lcp[rank[i]] = c;
		if (c)c--;
	}
	return lcp;
}
```

LIS binary Search

```
void LIS_binarySearch(vector<int> v) {
	int n = v.size();
	vector<int> last(n), prev(n, -1);
	int length = 0;
	auto BS = [&](int val) {
		int st = 1, ed = length, md, rt = length;
		while (st <= ed) {
			md = st + ed >> 1;
			if (v[last[md]] >= val)
				ed = md - 1, rt = md;
			else st = md + 1;
		}
		return rt;
	};
	for (int i = 1; i < n; i++) {
		if (v[i] < v[last[0]]) last[0] = i;
		else if (v[i] > v[last[length]]) {
			prev[i] = last[length];
			last[++length] = i;
		}
		else {
			int index = BS(v[i]);
			prev[i] = last[index - 1];
			last[index] = i;
		}
	}
	cout << length + 1 << "\n-\n";
	vector<int> out;
```

```
	for (int i = last[length]; i >= 0; i = prev[i])
		out.push_back(v[i]);
	reverse(out.begin(), out.end());
	for (auto it : out)cout << it << endl;
}
```

Bitmask

```
template<class Int>
bool getBit(Int num, int ind) { return ((num >> ind) & 1); }

template<class Int>
Int setBit(Int num, int ind, bool val) {
	return val ? (num | ((Int)(1) << ind)) : (num & ~((Int)(1) << ind));
}

template<class Int>
Int flipBit(Int num, int ind) { return (num ^ ((Int)(1) << ind)); }

template<class Int>
Int leastBit(Int num) { return (num & -num); }

//num%mod, mod is a power of 2
template<class Int>
Int Mod(Int num, Int mod) { return (num & mod - 1); }

template<class Int>
bool isPowerOfTwo(Int num) { return (num & num - 1) == 0; }

void genAllSubmask(int mask) {
	for (int subMask = mask;; subMask = (subMask - 1) & mask) {
		//code
		if (subMask == 0)break;
	}
}

// for run __builtin_popcount in visual
#ifdef _MSC_VER
#include <intrin.h>
#define __builtin_popcount __popcnt
#ifdef _WIN64
#define __builtin_popcountll __popcnt64
#else
inline int __builtin_popcountll(__int64 a) {
	return __builtin_popcount((unsigned int)a) + __builtin_popcount(a >> 32);
}
#endif
#endif
```

Sort

Merge sort

```
long long cnt = 0;
vector<int> v, temp;
void merge_sort(int s, int e) {
```

```
	if (s + 1 >= e) return;
	int m = s + (e - s >> 1);
	merge_sort(s, m);
	merge_sort(m, e);
	for (int i = s; i < e; i++) temp[i] = v[i];
	int i = s, j = m, k = s;
	while (i < m && j < e)
		if (temp[i] <= temp[j]) v[k++] = temp[i++];
		else  v[k++] = temp[j++], cnt += j - k;
	while (i < m)v[k++] = temp[i++];
	while (j < e)v[k++] = temp[j++];
}
```

Radix sort

```
// O(n*log(n)/log(base))
// O(n + base) memory
void radix_sort(vector<int>& v, int base) {
	vector<int> tmp(v.size());
	int p = 1;
	for (int it = 0; it < 10; it++, p *= base) {
		vector<int>  frq(base);
		for (auto& it : v)
			frq[(it / p) % base]++;
		for (int i = 1; i < base; i++)
			frq[i] += frq[i - 1];
		for (int i = v.size() - 1; i >= 0; i--)
			tmp[--frq[(v[i] / p) % base]] = v[i];
		v = tmp;
	}
}
```

Coordinate Compress

```
void coordinateCompress(vector<int>& axes, vector<int>& iToV,
	map<int, int>& vToI, int start = 2, int step = 2) {
	for (auto it : axes) vToI[it] = 0;
	iToV.resize(start + step * vToI.size());
	int idx = 0;
	for (auto& it : vToI) {
		it.second = start + step * idx;
		iToV[it.second] = it.first;
		idx++;
	}
}
```

Hash

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

```
		return x ^ (x >> 31);
	}
	// for pair
	size_t operator()(pair<uint64_t, uint64_t> x) const {
		static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
		return splitmix64(x.first + FIXED_RANDOM) ^ (splitmix64(x.second +
FIXED_RANDOM) >> 1);
	}
	// for single element
	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);
	}

};
```

Random number

```
//write this line once in top
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count()* ((uint64_t) new
char | 1));

// use this instead of rand()
long long rnd = uniform_int_distribution<long long>(low, high)(rng);
```

Java

Scanner

```
package other_algorithms;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

class Scanner
{
	StringTokenizer st;
	BufferedReader br;

	public Scanner(InputStream s){	br = new BufferedReader(new
InputStreamReader(s));}

	public String next() throws IOException
	{
		while (st == null || !st.hasMoreTokens())
			st = new StringTokenizer(br.readLine());
		return st.nextToken();
	}

	public int nextInt() throws IOException {return Integer.parseInt(next());}
```

```
	public long nextLong() throws IOException {return Long.parseLong(next());}

	public String nextLine() throws IOException {return br.readLine();}

	public double nextDouble() throws IOException
	{
		String x = next();
		StringBuilder sb = new StringBuilder("0");
		double res = 0, f = 1;
		boolean dec = false, neg = false;
		int start = 0;
		if(x.charAt(0) == '-')
		{
			neg = true;
			start++;
		}
		for(int i = start; i < x.length(); i++)
			if(x.charAt(i) == '.')
			{
				res = Long.parseLong(sb.toString());
				sb = new StringBuilder("0");
				dec = true;
			}
			else
			{
				sb.append(x.charAt(i));
				if(dec)
					f *= 10;
			}
		res += Long.parseLong(sb.toString()) / f;
		return res * (neg?-1:1);
	}

	public boolean ready() throws IOException {return br.ready();}
}
```

Segment tree

```
package data_structures.trees;

import java.util.Scanner;

// Range Sum Query (with lazy propagation)

public class SegmentTree {	// 1-based DS, OOP

	int N;			//the number of elements in the array as a power of
2 (i.e. after padding)
	int[] array, sTree, lazy;

	SegmentTree(int[] in)
	{
		array = in; N = in.length - 1;
```

```
		sTree = new int[N<<1];			//no. of nodes = 2*N - 1, we add one to
cross out index zero
		lazy = new int[N<<1];
		build(1,1,N);
	}

	void build(int node, int b, int e)	// O(n)
	{
		if(b == e)
			sTree[node] = array[b];
		else
		{
			int mid = b + e >> 1;
			build(node<<1,b,mid);
			build(node<<1|1,mid+1,e);
			sTree[node] = sTree[node<<1]+sTree[node<<1|1];
		}
	}

	void update_point(int index, int val)		// O(log n)
	{
		index += N - 1;
		sTree[index] += val;
		while(index>1)
		{
			index >>= 1;
			sTree[index] = sTree[index<<1] + sTree[index<<1|1];
		}
	}

	void update_range(int i, int j, int val)		// O(log n)
	{
		update_range(1,1,N,i,j,val);
	}

	void update_range(int node, int b, int e, int i, int j, int val)
	{
		if(i > e || j < b)
			return;
		if(b >= i && e <= j)
		{
			sTree[node] += (e-b+1)*val;
			lazy[node] += val;
		}
		else
		{
			int mid = b + e >> 1;
			propagate(node, b, mid, e);
			update_range(node<<1,b,mid,i,j,val);
			update_range(node<<1|1,mid+1,e,i,j,val);
			sTree[node] = sTree[node<<1] + sTree[node<<1|1];
		}

	}
	void propagate(int node, int b, int mid, int e)
```

```
	{
		lazy[node<<1] += lazy[node];
		lazy[node<<1|1] += lazy[node];
		sTree[node<<1] += (mid-b+1)*lazy[node];
		sTree[node<<1|1] += (e-mid)*lazy[node];
		lazy[node] = 0;
	}

	int query(int i, int j)
	{
		return query(1,1,N,i,j);
	}

	int query(int node, int b, int e, int i, int j)		// O(log n)
	{
		if(i>e || j <b)
			return 0;
		if(b>= i && e <= j)
			return sTree[node];
		int mid = b + e >> 1;
		propagate(node, b, mid, e);
		int q1 = query(node<<1,b,mid,i,j);
		int q2 = query(node<<1|1,mid+1,e,i,j);
		return q1 + q2;

	}
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int N = 1; while(N < n) N <<= 1; //padding

		int[] in = new int[N + 1];
		for(int i = 1; i <= n; i++)
			in[i] = sc.nextInt();

		sc.close();
	}

}
```

Leap Year

```
bool isLeap(int y) {
	return y % 400 == 0 || (y % 100 != 0 && y % 4 == 0);
}
string dayOfTheWeek(int day, int month, int year) {
	vector<int> md = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
	vector<string> show{ "Friday", "Saturday", "Sunday", "Monday",
		"Tuesday", "Wednesday", "Thursday" };
	int idx = 6;
	for (int y = 1971; y < year; y++)
		idx += (isLeap(y) ? 366 : 365);
	for (int m = 1; m < month; m++)
		idx += (isLeap(year) && m == 2 ? 29 : md[m]);
	idx += day;
	return show[idx % 7];}
```