**DS: Li Chao Tree**

template<typename data\_t>

struct Line {

data\_t a, b;

Line() : a(0), b(-inf) {}

Line(data\_t a, data\_t b) : a(a), b(b) {}

data\_t get(data\_t x) {

return a \* x + b;

}void add(Line x) {

a += x.a;

b += x.b;

}};

struct Node {

Line<data\_t> line = Line<data\_t>();

Line<data\_t> lazy = Line<data\_t>(0, 0);

Node \*lc = nullptr;

Node \*rc = nullptr;

void apply(data\_t l, data\_t r, Line<data\_t> v) {

line.add(v);

lazy.add(v);

}

};void PushLazy(Node\* &n, data\_t tl, data\_t tr) {

if (n == nullptr) return;

if (n->lc == nullptr) n->lc = new Node();

if (n->rc == nullptr) n->rc = new Node();

data\_t mid = (tl + tr) / 2;

n->lc->apply(tl, mid, n->lazy);

n->rc->apply(mid + 1, tr, n->lazy);

n->lazy = Line<data\_t>(0, 0);

}

void PushLine(Node\* &n, data\_t tl, data\_t tr) {

if (n == nullptr) return;

data\_t mid = (tl + tr) / 2;

InsertLineKnowingly(n->lc, tl, mid, n->line);

InsertLineKnowingly(n->rc, mid + 1, tr, n->line);

n->line = Line<data\_t>();

}

void InsertLineKnowingly(Node\* &n, data\_t tl, data\_t tr, Line<data\_t> x) {

if (n == nullptr) n = new Node();

if (n->line.get(tl) < x.get(tl)) swap(n->line, x);

if (n->line.get(tr) >= x.get(tr)) return;

if (tl == tr) return;

data\_t mid = (tl + tr) / 2;

PushLazy(n, tl, tr);

if (n->line.get(mid) > x.get(mid)) {

InsertLineKnowingly(n->rc, mid + 1, tr, x);

} else {

swap(n->line, x);

InsertLineKnowingly(n->lc, tl, mid, x);

}

}

void InsertLine(Node\* &n, data\_t tl, data\_t tr, data\_t l, data\_t r, Line<data\_t> x) {

if (tr < l || r < tl || tl > tr || l > r) return;

if (n == nullptr) n = new Node();

if (l <= tl && tr <= r) return InsertLineKnowingly(n, tl, tr, x);

data\_t mid = (tl + tr) / 2;

PushLazy(n, tl, tr);

InsertLine(n->lc, tl, mid, l, r, x);

InsertLine(n->rc, mid + 1, tr, l, r, x);

}

void AddLine(Node\* &n, data\_t tl, data\_t tr, data\_t l, data\_t r, Line<data\_t> x) {

if (tr < l || r < tl || tl > tr || l > r) return;

if (n == nullptr) n = new Node();

if (l <= tl && tr <= r) return n->apply(tl, tr, x);

data\_t mid = (tl + tr) / 2;

PushLazy(n, tl, tr);

PushLine(n, tl, tr);

AddLine(n->lc, tl, mid, l, r, x);

AddLine(n->rc, mid + 1, tr, l, r, x);

}

data\_t Query(Node\* &n, data\_t tl, data\_t tr, data\_t x) {

if (n == nullptr) return -inf;

if (tl == tr) return n->line.get(x);

data\_t res = n->line.get(x);

data\_t mid = (tl + tr) / 2;

PushLazy(n, tl, tr);

if (x <= mid) {

res = max(res, Query(n->lc, tl, mid, x));

} else {

res = max(res, Query(n->rc, mid + 1, tr, x));

}

return res;

}

void InsertLine(data\_t l, data\_t r, Line<data\_t> x) {

return InsertLine(root, 0, sz - 1, l, r, x);

}

void AddLine(data\_t l, data\_t r, Line<data\_t> x) {

return AddLine(root, 0, sz - 1, l, r, x);

}

data\_t Query(data\_t x) {

return Query(root, 0, sz - 1, x);

}

**Persistent Segment Tree**

int st[50\*300002], lftchild[50\*300002], rgtchild[50\*300002],lxy[50\*300002] id,/\*nodes\*/, n, m;

vector<int> roots;

int p[mxn];

int build(int l, int r) {

int root = id++;

if(l==r) {

st[root] = 0;

return root;

}

int m= (l+r)/2;

lftchild[root] =build(l,m);

rgtchild[root] = build(m+1, r);

st[root] = st[lftchild[root]] + st[rgtchild[root]];

return root;

}

int updt(int previousroot, int l, int r, int s, int t, int v) {

if(t<l || r<s) return previousroot;

int root = id++;

lxy[root] = lxy[previousroot];

if(s<=l && r<=t) {

st[root] = st[previousroot]+v\*(r-l+1);

lxy[root] = lxy[previousroot]+v;

lftchild[root] = lftchild[previousroot];

rgtchild[root] = rgtchild[previousroot];

return root;

}

int m = (l+r)/2;

int c = updt(lftchild[previousroot], l, m, s, t, v);

int d = updt(rgtchild[previousroot], m+1, r, s, t, v);

st[root] = st[c]+st[d]+lxy[root]\*(r-l+1);

lftchild[root] = c;

rgtchild[root] = d;

return root;

}

int \_find(int root, int l, int r, int s, int t) {

if(r<s || t<l) return 0;

if(s<=l && r<=t) return st[root];

int m = (l+r)/2;

return (min(r, t)-max(l, s)+1)\*lxy[root]+\_find(lftchild[root], l, m, s, t)+\_find(rgtchild[root], m+1, r, s, t);

}

int main(){

roots[0] = build(1, m);

}

**WaveletTree**

**Description: fast (k-th smallest value, count of ≤ k values, sum of**

≤ k values) in range

const int MAX = 100005;

// if array elements are big (1e9), compress first

// after construction, ara will be changed

int ara[MAX];

struct wavelet\_tree {

int lo, hi;

wavelet\_tree \*l, \*r;

vector<int> b;

vector<ll> c; // prefix sum of elements for sum query

// elements are in [x,y], indices are [from, to)

wavelet\_tree(int \*from, int \*to, int x, int y) {

lo = x, hi = y;

if (from >= to) return;

if (hi == lo) {

b.reserve(to - from + 1);

b.push\_back(0);

c.reserve(to - from + 1);

c.push\_back(0);

for (auto it = from; it != to; ++it) {

b.push\_back(b.back() + 1);

c.push\_back(c.back() + rm[\*it]);

}

return;

}

int mid = (lo + hi) / 2;

auto f = [mid](int x) {

return x <= mid;

};

b.reserve(to - from + 1);

b.push\_back(0);

c.reserve(to - from + 1);

c.push\_back(0);

for (auto it = from; it != to; it++) {

b.push\_back(b.back() + f(\*it));

c.push\_back(c.back() + rm[\*it]);

}

auto pivot = stable\_partition(from, to, f);

l = new wavelet\_tree(from, pivot, lo, mid);

r = new wavelet\_tree(pivot, to, mid + 1, hi);

}

// k-th smallest element in subarray [l,r]

int kth(int l, int r, int k) {

if (l > r) return 0;

if (lo == hi) return lo;

int inLeft = b[r] - b[l - 1];

int lb = b[l - 1];

int rb = b[r];

if (k <= inLeft) return this->l->kth(lb + 1, rb, k);

return this->r->kth(l - lb, r - rb, k - inLeft);

}

// number of elements <= k in subarray [l,r]

int LTE(int l, int r, int k) {

if (l > r or k < lo) return 0;

if (hi <= k) return r - l + 1;

int lb = b[l - 1], rb = b[r];

return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l -lb, r - rb, k);

}

// number of occurrences of k in subarray [l,r]

int count(int l, int r, int k) {

if (l > r or k<lo or k> hi) return 0;

if (lo == hi) return r - l + 1;

int lb = b[l - 1], rb = b[r], mid = (lo + hi) / 2;

if (k <= mid) return this->l->count(lb + 1, rb, k);

return this->r->count(l - lb, r - rb, k);

}

// sum of elements <= k in subarray [l,r]

ll sumk(int l, int r, int k) {

if (l > r or k < lo) return 0;

if (hi <= k) return c[r] - c[l - 1];

int lb = b[l - 1], rb = b[r];

return this->l->sumk(lb + 1, rb, k) + this->r->sumk(l -lb, r - rb, k);

}

~wavelet\_tree() {

delete l;

delete r;

}

};

int main() {

int n;

cin >> n;

for (int i = 1; i <= n; i++) cin >> ara[i];

wavelet\_tree \*Tree = new wavelet\_tree(ara + 1, ara + n +1, 1, 1e5);

}

**Disjoint Spase Table O(NlgN) O(1)**

int mod = 1e9 + 7;

struct DST {

vector<vector<int>> left, right;

int k, n;

DST(vector<int> & a) {

n = (int)a.size();

k = log2(n) + 2;

left.assign(k + 1, vector<int>(n));

right.assign(k + 1, vector<int>(n));

for(int j = 0; (1 << j) <= n; ++j) {

int mask = (1 << j) - 1;

int nw = 1; //neutral

for(int i = 0; i < n; ++i) {

nw = 1LL \* nw \* a[i] % mod;//prefix value

left[j][i] = nw;

if((i & mask) == mask) nw = 1; //neutral

}

nw = 1; //neutral

for(int i = n - 1; i >= 0; --i) {

nw = 1LL \* nw \* a[i] % mod;//prefix value

right[j][i] = nw;

if((i & mask) == 0) nw = 1; //neutral

}

}

}

int query(int l, int r) {

if(l == r) return left[0][l];

int i = 31 - \_\_builtin\_clz(l ^ r);

int uno = left[i][r];

int dos = right[i][l];

return 1LL \* uno \* dos % mod;

}

};

**Interval Set**

//for Q assign operation it takes Qlogn time in total

template<class T>

struct interval\_set {

map<pair<int, int>, T> value;//{r,l}=val

void init(int n) {

value[ {n, 1}] = (T)0; //initial value

}

//assign a[i]=val for l<=i<=r

//returns affected ranges before performing this assign operation

vector<pair<pair<int, int>, T> > assign(int l, int r, T val) {

auto bg = value.lower\_bound({l, 0})->first;

if(bg.second != l) {

T val = value[bg];

value.erase(bg);

value[ {l - 1, bg.second}] = val;

value[ {bg.first, l}] = val;

}

auto en = value.lower\_bound({r, 0})->first;

if(en.first != r) {

T val = value[en];

value.erase(en);

value[ {en.first, r + 1}] = val;

value[ {r, en.second}] = val;

}

vector<pair<pair<int, int>, T> > ret;

auto itt = value.lower\_bound({l, 0});

while(true) {

if(itt == value.end() || itt->first.first > r) break;

ret.push\_back({{itt->first.second, itt->first.first}, itt->second});

++itt;

}

for(auto it : ret)

value.erase({it.first.second, it.first.first});

value[ {r, l}] = val;

return ret;

}

};

**Iterative Segment Tree**

const int inf = 0x3f3f3f3f, N = 3e5;

struct RMQ {

vector <int> t;

int n;

RMQ(int n) : n(n) {

t.resize(n << 1);

}

void build(int a[]) {

for(int i = 0; i < n; i++) t[n + i] = a[i];

for(int i = n - 1; i > 0; --i) t[i] = min(t[i << 1], t[i << 1 | 1]);

}

void modify(int p, int v) {

for(t[p += n] = v; p > 1; p >>= 1) t[p >> 1] = min(t[p], t[p ^ 1]);

}

int query(int l, int r) {

int res = inf;

for(l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {

if(l & 1) res = min(res, t[l++]);

if(r & 1) res = min(res, t[--r]);

}

return res;

}

};

**DSU on Tree:**

int cnt[maxn];

void dfs(int v, int p, bool keep){

int mx = -1, bigChild = -1;

for(auto u : g[v])

if(u != p && sz[u] > mx)

mx = sz[u], bigChild = u;

for(auto u : g[v])

if(u != p && u != bigChild)

dfs(u, v, 0);

if(bigChild != -1)

dfs(bigChild, v, 1);

for(auto u : g[v])

if(u != p && u != bigChild)

for(int p = st[u]; p < ft[u]; p++)

cnt[ col[ ver[p] ] ]++;

cnt[ col[v] ]++;

if(keep == 0)

for(int p = st[v]; p < ft[v]; p++)

cnt[ col[ ver[p] ] ]--;

}

**HLD**

vector<int>adj[N];

int curPos,depth[N],headofCurrentChain[N],

heavyChild[N],parent[N],pos[N];

int DFS(int cur) {

int childSize,size=1,maxChildSize=0;

for(int x : adj[cur]) {

if(x ^ parent[cur]) {

parent[x]=cur,depth[x]=depth[cur]+1;

childSize=DFS(x),size+=childSize;

if(childSize > maxChildSize) {

heavyChild[cur]=x;

maxChildSize=childSize;

}

}

}

return size;

}

void Decompose(int cur,int headNode) {

pos[cur]=++curPos;

headofCurrentChain[cur]=headNode;

if(heavyChild[cur])Decompose(heavyChild[cur],headNode);

for(int x : adj[cur]) {

if(x != parent[cur] && x != heavyChild[cur])Decompose(x,x);

}

}

void HeavyLightDecomposition(const int& n) {

DFS(1),curPos=0,Decompose(1,1);

for(i=1; i<=n; i++)val[pos[i]]=a[i];

}

long long HLDQuery(int x,int y) {

long long ans=0,curVal;

while(headofCurrentChain[x] != headofCurrentChain[y]) {

if(depth[headofCurrentChain[x]] > depth[headofCurrentChain[y]])swap(x,y);

curVal=SegmentTreeQuery(1,1,n,pos[headofCurrentChain[y]],pos[y]);

ans+=curVal;

y=parent[headofCurrentChain[y]];

}

if(depth[x] > depth[y])swap(x,y);

curVal=SegmentTreeQuery(1,1,n,pos[x],pos[y]);

return ans+curVal;

}

**Merge Sort:**

long long ans,sum[22][N];

int n,a[N],sorted[N],tree[22][N],link[22][N];

void Build(int l,int r,int lvl) {

if(l == r)return;

int i,equalCnt=0,lessCnt=0,mid=(l+r) >> 1,lChild=l,rChild=mid+1;

for(i=l; i<=r; i++) {

if(tree[lvl][i] < sorted[mid])++lessCnt;

}

for(i=l; i<=r; i++) {

if((tree[lvl][i] < sorted[mid]) || (tree[lvl][i] == sorted[mid] && equalCnt < (mid-l-lessCnt+1))) {

tree[lvl+1][lChild++]=tree[lvl][i];

sum[lvl][i]=sum[lvl][i-1]+tree[lvl][i];

if(tree[lvl][i] == sorted[mid])++equalCnt;

} else {

tree[lvl+1][rChild++]=tree[lvl][i];

sum[lvl][i]=sum[lvl][i-1];

}

link[lvl][i]=link[lvl][l-1]+lChild-l;

}

Build(l,mid,lvl+1), Build(mid+1,r,lvl+1);

}

void BuildTree() {

for(int i=1; i<=n; i++)tree[0][i]=sorted[i]=a[i];

sort(sorted+1,sorted+n+1),Build(1,n,0);

}

//ans stores the sum of first k smallest elements

int kth(int start,int end,int curLvl,int l,int r,int k) {

if(start == end) {

ans+=tree[curLvl][start];

return tree[curLvl][start];

}

int cnt=link[curLvl][r]-link[curLvl][l-1],mid=(start+end) >> 1;

if(cnt >= k) {

int newL=link[curLvl][l-1]-link[curLvl][start-1];

int newR=link[curLvl][r]-link[curLvl][start-1];

return kth(start,mid,curLvl+1,start+newL,start+newR-1,k);

} else {

int newL=l-start-(link[curLvl][l-1]-link[curLvl][start-1]);

int newR=r-start+1-(link[curLvl][r]-link[curLvl][start-1]);

ans+=sum[curLvl][r]-sum[curLvl][l-1];

return kth(mid+1,end,curLvl+1,mid+1+newL,mid+newR,k-cnt);

}

}

//ans stores the sum of elements less equal k

int LessEqualK(int start,int end,int curLvl,int l,int r,int k) {

if(start == end) {

if(start > r || end < l)return 0;

ans+=tree[curLvl][start]\*(tree[curLvl][start] <= k);

return tree[curLvl][start] <= k;

}

int cnt=link[curLvl][r]-link[curLvl][l-1],mid=(start+end) >> 1;

if(sorted[mid] <= k) {

ans+=sum[curLvl][r]-sum[curLvl][l-1];

int newL=l-start-(link[curLvl][l-1]-link[curLvl][start-1]);

int newR=r-start+1-(link[curLvl][r]-link[curLvl][start-1]);

return cnt+LessEqualK(mid+1,end,curLvl+1,mid+1+newL,mid+newR,k);

} else {

int newL=link[curLvl][l-1]-link[curLvl][start-1];

int newR=link[curLvl][r]-link[curLvl][start-1];

return LessEqualK(start,mid,curLvl+1,start+newL,start+newR-1,k);

}

}

**MO,s with Update:**

const int magic = 2143;

int lastVal[M];

bool visited[M];

struct Query {

int id,l,r,updateCnt;

Query() {}

Query(int \_l,int \_r,int \_id,int \_updateCnt) {

l=\_l, r=\_r, id=\_id, updateCnt=\_updateCnt;

} bool operator < (const Query& q) {

if(l/magic == q.l/magic) {

if(r/magic == q.r/magic)return ((r/magic) & 1) ? (updateCnt > q.updateCnt) : (updateCnt < q.updateCnt);

return r/magic < q.r/magic;

}

return l/magic < q.l/magic;

}

} queries[M];

struct Update {

int id,prevVal,val;

Update() {}

Update(int \_id,int \_prevVal,int \_val) {

id=\_id, prevVal=\_prevVal, val=\_val;

}

} updates[M];

int a[M],ans[M],curAns,freq[M];

void Increment(int x) {

++freq[x];

if(freq[x] == 1)++curAns;

}

void Decrement(int x) {

--freq[x];

if(!freq[x])--curAns;

}

void Upd(int x) {

int y = a[x];

if(!visited[x])Increment(y);

else Decrement(y);

visited[x] = 1-visited[x];

}

void Add(int i) {

int id=updates[i].id, x=updates[i].val;

if(visited[id])Decrement(a[id]);

a[id] = x;

if(visited[id])Increment(a[id]);

}

void Delete(int i) {

int id=updates[i].id, x=updates[i].prevVal;

if(visited[id])Decrement(a[id]);

a[id] = x;

if(visited[id])Increment(a[id]);

}

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

int n,q;

cin >> n >> q;

for(int i=1; i<=n; i++)cin >> a[i], lastVal[i]=a[i];

int queryCnt=0, updateCnt=0;

while(q--) {

int t;

cin >> t;

if(t == 1) {

int l,r;

cin >> l >> r;

++queryCnt;

queries[queryCnt] = Query(l, r, queryCnt, updateCnt);

} else {

int i,x;

cin >> i >> x;

updates[++updateCnt] = Update(i, lastVal[i], x);

lastVal[i] = x;

}

}

sort(queries+1, queries+queryCnt+1);

int l=1, r=1, updatesProcessed=0;

for(int i=1; i<=queryCnt; i++) {

while(queries[i].updateCnt < updatesProcessed)Delete(updatesProcessed--);

while(queries[i].updateCnt > updatesProcessed)Add(++updatesProcessed);

while(queries[i].l < l)Upd(--l);

while(queries[i].r >= r)Upd(r++);

while(queries[i].l > l)Upd(l++);

while(queries[i].r+1 < r)Upd(--r);

ans[queries[i].id] = curAns;

}

for(int i=1; i<=queryCnt; i++)cout << ans[i] << '\n';

}

**Treap**

mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

struct treap {

struct node {

int value = 0, priority = rng(), size = 1;

int sum = 0, pro = 0;

node \*l = NULL, \*r = NULL;

node(int x) : value(x) {}

} \*root = NULL;

int size(node \*v) {

return !v ? 0 : v -> size;

}

int op(node \*v) {

if(v==NULL) return 0;

return v->sum;

}

void recalc(node \*v) {

if(!v) return;

v -> size = size(v -> l) + size(v -> r) + 1;

v->sum = op(v->l)+op(v->r)+v->value;

}

void prop(node \* v) {

if(v==NULL) return;

// v->value+=v->pro;

if(v->pro) swap(v->l, v->r);

if(v->l) v->l->pro^=v->pro, recalc(v->l);

if(v->r) v->r->pro^=v->pro, recalc(v->r);

v->pro = 0;

recalc(v);

}

node \*merge(node \*p, node \*q) {

prop(p), prop(q);

if(!p || !q) return !p ? q : p;

if(p -> priority < q -> priority) {

p -> r = merge(p -> r, q);

recalc(p);

return p;

} else {

q -> l = merge(p, q -> l);

recalc(q);

return q;

}

}

pair <node \*, node \*> split(node \*v, int cnt) {

prop(v);

if(!v) return {NULL, NULL};

node \* p, \* q;

if(size(v -> l) >= cnt) {

tie(p, q) = split(v -> l, cnt);

v -> l = q;

recalc(v);

return {p, v};

} else {

tie(p, q) = split(v -> r, cnt - size(v -> l) - 1);

v -> r = p;

recalc(v);

return {v, q};

}

}

void insert(int x) {

root = merge(root, new node(x));

}

void dfs(node\* p) {

prop(p);

if(p==NULL) return;

dfs(p->l);

cout << char(p->value)<<"";

dfs(p->r);

return ;

}

};

void upd(treap &t, int f, int l, int vl) {

auto lst = t.split(t.root, l); // rng upd

auto fst = t.split(lst.first, f-1);

fst.second-> pro^=vl;

t.merge(fst.first, fst.second);

t.merge(lst.first, lst.second);

}

int \_F(treap &t, int f, int l) {

auto lst = t.split(t.root, l);// rng \_f

auto fst = t.split(lst.first, f-1);

int res = t.op(fst.second);

t.merge(fst.first, fst.second);

t.merge(lst.first, lst.second);

return res;

}

**Dynamic Programming**: **LCHT**

using ii = pair <int, int>;

long double intersect(ii &a, ii &b) {

return (long double)(a.second - b.second) / (b.first - a.first);

}

struct CHT {

deque <ii> dq;

void add(ii p) {

while(dq.size() >= 2 and intersect(dq[dq.size() - 2], dq.back()) >= intersect(dq[dq.size() - 2], p))

dq.pop\_back();

dq.push\_back(p);

}

int query(int x) {

while(dq.size() >= 2 and dq[0].first \* x + dq[0].second >= dq[1].first \* x + dq[1].second)

dq.pop\_front();

return dq[0].first \* x + dq[0].second;

}

void clear() {

while(dq.size()) dq.pop\_back();

}

};

**D&C DP**



//C(a,c) + C(b,d) <= C(a,d) + C(b,c) where a <= b <= c <= d

int m, n;

vector<long long> dp\_before(n), dp\_cur(n);

long long C(int i, int j);

// compute dp\_cur[l], ... dp\_cur[r] (inclusive)

void compute(int l, int r, int optl, int optr) {

if (l > r) return;

int mid = (l + r) >> 1;

pair<long long, int> best = {LLONG\_MAX, -1};

for (int k = optl; k <= min(mid, optr); k++) {

best = min(best, {(k ? dp\_before[k - 1] : 0) + C(k, mid), k});

}

dp\_cur[mid] = best.first;

int opt = best.second;

compute(l, mid - 1, optl, opt);

compute(mid + 1, r, opt, optr);

}

int solve() {

for (int i = 0; i < n; i++)

dp\_before[i] = C(0, i);

for (int i = 1; i < m; i++) {

compute(0, n - 1, 0, n - 1);

dp\_before = dp\_cur;

}

return dp\_before[n - 1];

}

**DCHT**



typedef long long LL;

struct Line {

mutable LL m, b, p;

bool operator<(const Line &o) const {

return m < o.m;

} bool operator<(const LL &x) const {

return p < x;

}

};

struct LineContainer : multiset<Line, less<>> { // for min query add(-m, -b); -query(x);

// (for doubles, use inf = 1/.0, div(a,b) = a/b) const LL inf = LLONG\_MAX;

LL div(LL a, LL b) {

return a / b - ((a ^ b) < 0 && a % b);

} bool isect(iterator x, iterator y) {

if (y == end()) {

x->p = inf;

return false;

}

if (x->m == y->m)

x->p = x->b > y->b ? inf : -inf;

else

x->p = div(y->b - x->b, x->m - y->m);

return x->p >= y->p;

}

void add(LL m, LL b) {

auto z = insert({m, b, 0}), y = z++, x = y;

while (isect(y, z))

z = erase(z);

if (x != begin() && isect(--x, y)) isect(x, y = erase(y));

while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));

}

LL query(LL x) {

assert(!empty());

auto l = \*lower\_bound(x);

return l.m \* x + l.b;

}

};

**Kunth DP**



int solve() {

int N;

int dp[N][N], opt[N][N];

auto C = [&](int i, int j) {

};

for (int i = 0; i < N; i++) {

opt[i][i] = i;

}

for (int i = N-2; i >= 0; i--) {

for (int j = i+1; j < N; j++) {

int mn = INT\_MAX;

int cost = C(i, j);

for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j]); k++) {

if (mn >= dp[i][k] + dp[k+1][j] + cost) {

opt[i][j] = k;

mn = dp[i][k] + dp[k+1][j] + cost;

}

}

dp[i][j] = mn;

}

}

cout << dp[0][N-1] << endl;

}

**Graph**: **Articulation point**

const int mxn = 20;

int ap[mxn], disc[mxn], low[mxn];

vector<int> adj[mxn];

void dfsAP(int u, int p) {

int children = 0;

low[u] = disc[u] = ++Time;

for (int& v : adj[u]) {

if (v == p) continue;

if (!disc[v]) {

children++;

dfsAP(v, u);

if (disc[u] <= low[v]) // use < for bridge

ap[u] = 1;

low[u] = min(low[u], low[v]);

} else

low[u] = min(low[u], disc[v]);

}

if(children>2 && u==root) ap[u] = 1; //don’t use bridge

return;

}

void AP() {

ap = low = disc = vector<int>(adj.size());

Time = 0;

for (int u = 0; u < adj.size(); u++)

if (!disc[u])

dfsAP(u, u);

}

**Online Bridge**

vector<int> par, dsu\_2ecc, dsu\_cc, dsu\_cc\_size;

int bridges;

int lca\_iteration;

vector<int> last\_visit;

void init(int n) {

par.resize(n);

dsu\_2ecc.resize(n);

dsu\_cc.resize(n);

dsu\_cc\_size.resize(n);

lca\_iteration = 0;

last\_visit.assign(n, 0);

for (int i=0; i<n; ++i) {

dsu\_2ecc[i] = i;

dsu\_cc[i] = i;

dsu\_cc\_size[i] = 1;

par[i] = -1;

}

bridges = 0;

}

int find\_2ecc(int v) {

if (v == -1)

return -1;

return dsu\_2ecc[v] == v ? v : dsu\_2ecc[v] = find\_2ecc(dsu\_2ecc[v]);

}

int find\_cc(int v) {

v = find\_2ecc(v);

return dsu\_cc[v] == v ? v : dsu\_cc[v] = find\_cc(dsu\_cc[v]);

}

void make\_root(int v) {

v = find\_2ecc(v);

int root = v;

int child = -1;

while (v != -1) {

int p = find\_2ecc(par[v]);

par[v] = child;

dsu\_cc[v] = root;

child = v;

v = p;

}

dsu\_cc\_size[root] = dsu\_cc\_size[child];

}

void merge\_path (int a, int b) {

++lca\_iteration;

vector<int> path\_a, path\_b;

int lca = -1;

while (lca == -1) {

if (a != -1) {

a = find\_2ecc(a);

path\_a.push\_back(a);

if (last\_visit[a] == lca\_iteration) {

lca = a;

break;

}

last\_visit[a] = lca\_iteration;

a = par[a];

}

if (b != -1) {

b = find\_2ecc(b);

path\_b.push\_back(b);

if (last\_visit[b] == lca\_iteration) {

lca = b;

break;

}

last\_visit[b] = lca\_iteration;

b = par[b];

}

}

for (int v : path\_a) {

dsu\_2ecc[v] = lca;

if (v == lca)

break;

--bridges;

}

for (int v : path\_b) {

dsu\_2ecc[v] = lca;

if (v == lca)

break;

--bridges;

}

}

void add\_edge(int a, int b) {

a = find\_2ecc(a);

b = find\_2ecc(b);

if (a == b)

return;

int ca = find\_cc(a);

int cb = find\_cc(b);

if (ca != cb) {

++bridges;

if (dsu\_cc\_size[ca] > dsu\_cc\_size[cb]) {

swap(a, b);

swap(ca, cb);

}

make\_root(a);

par[a] = dsu\_cc[a] = b;

dsu\_cc\_size[cb] += dsu\_cc\_size[a];

} else {

merge\_path(a, b);

}

}

**BCC**

vector <int> adj[N];

int dis[N], low[N], col[N], ins[N], t, id;

stack <int> stk;

void BCC(int u, int p) {

dis[u] = low[u] = ins[u] = ++t;

stk.push(u);

for(int v: adj[u]) {

if(!dis[v]) {

BCC(v, u);

low[u] = min(low[u], low[v]);

} else if(ins[v] && p != v)

low[u] = min(low[u], dis[v]);

}

if(dis[u] == low[u]) {

++id;

int v;

do {

v = stk.top();

stk.pop();

ins[v] = 0;

col[v] = id;

} while(v != u);

}

}

**Dynamic Connectivity**

struct DSU{

vector <int> p, sz, stk;

DSU(int n) {

p.resize(n + 1); sz.resize(n + 1, 1);

for(int i = 1; i <= n; i++) p[i] = i;

}

int find(int x) {

return (p[x] == x) ? p[x] : find(p[x]);

}

int merge(int x, int y) {

if((x = find(x)) ^ (y = find(y))) {

if(sz[x] > sz[y]) swap(x, y);

p[x] = y;

sz[y] += sz[x];

stk.push\_back(x);

return 1;

}

return 0; }

void roll\_back(int t) {

while(stk.size() > t) {

int x = stk.back();

stk.pop\_back();

sz[p[x]] -= sz[x];

p[x] = x;

} }

};

void update(int v, int l, int r, int L, int R, pair <int, int> e) {

if(l > R || r < L) return;

if(l >= L && r <= R) {

edges[v].push\_back(e);

return;

}

int mid = (l + r) / 2;

update(v \* 2, l, mid, L, R, e);

update(v \* 2 + 1, mid + 1, r, L, R, e);

}

void build(int v, int l, int r, DSU &d) {

int tm = d.stk.size();

for(auto e: edges[v]) d.merge(e.first, e.second);

if(l == r) {

for(auto Q: query[l]) ans[Q.second] = d.sz[d.find(Q.first)];

} else {

int mid = (l + r) / 2;

build(v \* 2, l, mid, d);

build(v \* 2 + 1, mid + 1, r, d);

}

d.roll\_back(tm);

}

**SAPA**

const int INF = 1000000000;

vector<vector<pair<int, int>>> adj;

bool spfa(int s, vector<int>& d) {

int n = adj.size();

d.assign(n, INF);

vector<int> cnt(n, 0);

vector<bool> inqueue(n, false);

queue<int> q;

d[s] = 0;

q.push(s);

inqueue[s] = true;

while (!q.empty()) {

int v = q.front();

q.pop();

inqueue[v] = false;

for (auto edge : adj[v]) {

int to = edge.first;

int len = edge.second;

if (d[v] + len < d[to]) {

d[to] = d[v] + len;

if (!inqueue[to]) {

q.push(to);

inqueue[to] = true;

cnt[to]++;

if (cnt[to] > n)

return false; // negative cycle

}

}

}

}

return true;

}

SCC:

vector<vector<int>> adj, adj\_rev;

vector<bool> used;

vector<int> order, component;

void dfs1(int v) {

used[v] = true;

for (auto u : adj[v])

if (!used[u])

dfs1(u);

order.push\_back(v);

}

void dfs2(int v) {

used[v] = true;

component.push\_back(v);

for (auto u : adj\_rev[v])

if (!used[u])

dfs2(u);

}

int main() {

int n;

for (;;) {

int a, b;

adj[a].push\_back(b);

adj\_rev[b].push\_back(a);

}

used.assign(n, false);

for (int i = 0; i < n; i++)

if (!used[i])

dfs1(i);

used.assign(n, false);

reverse(order.begin(), order.end());

for (auto v : order)

if (!used[v]) {

dfs2 (v);

component.clear();

}

}

**Tree hash**

#define int long long int

const int b[2] = {179, 191};

const int mod[2] = {916969619, 999999937};

vector<int> pw[2];// Initialije value

vector<int> siz;// Reisje

vector<pair<int, int>> myHash;//resije

vector<vector<int>> g;

void calc(int u, int f = -1) {

siz[u] = 1;

vector<pair<pair<int, int>, int>> subs;

for (int v : g[u]) if (v != f) {

calc(v, u);

subs.push\_back({myHash[v], v});

siz[u] += siz[v];

}

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

myHash[u] = {1, 1}; // leave node hsh vlue

for (auto [h, v] : subs) {

myHash[u].first \*= pw[0][siz[v] + siz[v]]; // cur = cur \* pw^2childsije + childhsh

myHash[u].first %= mod[0];

myHash[u].first += h.first;

if (myHash[u].first >= mod[0]) myHash[u].first -= mod[0];

myHash[u].second \*= pw[1][siz[v] + siz[v]];

myHash[u].second %= mod[1];

myHash[u].second += h.second;

if (myHash[u].second >= mod[1]) myHash[u].second -= mod[1];

}

myHash[u].first <<= 1;

if (myHash[u].first >= mod[0]) myHash[u].first -= mod[0];

myHash[u].first += 2;

if (myHash[u].first >= mod[0]) myHash[u].first -= mod[0];

myHash[u].second <<= 1;

if (myHash[u].second >= mod[1]) myHash[u].second -= mod[1];

myHash[u].second += 2;

if (myHash[u].second >= mod[1]) myHash[u].second -= mod[1];

}

**Eular Graph**

/\*euler path: starts from any vertex, visits every edge exactly once.

euler circuit: starts from any vertex, visits every edge exactly once and

returns to the start vertex.

for directed graph.\*/

int in[N+2],out[N+2];

vector<int>cir;//circuit

bool eulerCircuit(int n,int start) {

for(int i=1; i<=n; i++)if(in[i]!=out[i])return false;

stack<int>st;

st.push(start);

while(!st.empty()) {

int u=st.top();

if(adj[u].size()==0) {

cir.push\_back(u);

st.pop();

} else {

st.push(adj[u].back());

adj[u].pop\_back();

}

}

reverse(cir.begin(),cir.end());

return true;

}

void solve(int n) { //adj graph is given

for(int i=1; i<=n; i++) {

for(auto x:adj[i])out[i]++,in[x]++;

}

eulerCircuit(n,1);

}

**Flow**: **Dinic O(V\*V\*E)**

struct edge {

int u, v;

long long capacity, flow = 0;

edge(int u, int v, long long capacity) : u(u), v(v), capacity(capacity) {}

};

struct Dinic {

int s, t, id = 0, n;

const long long INF = 0x3f3f3f3f3f3f3f3f;

vector <edge> edges;

vector <vector <int> > adj;

vector <int> lvl, ptr;

queue <int> Q;

Dinic(int n, int s, int t) : n(n), s(s), t(t) {

adj.resize(n);

lvl.resize(n);

ptr.resize(n);

}

void addEdge(int u, int v, long long capacity) {

edges.push\_back(edge(u, v, capacity));

edges.push\_back(edge(v, u, 0));

adj[u].push\_back(id);

adj[v].push\_back(id + 1);

id += 2;

}

int bfs() {

fill(lvl.begin(), lvl.end(), -1);

fill(ptr.begin(), ptr.end(), 0);

Q.push(s);

lvl[s] = 0;

while(!Q.empty()) {

int u = Q.front();

Q.pop();

for(int i = 0; i < adj[u].size(); i++) {

if(lvl[edges[adj[u][i]].v] != -1 || edges[adj[u][i]].capacity - edges[adj[u][i]].flow == 0)

continue;

lvl[edges[adj[u][i]].v] = lvl[u] + 1;

Q.push(edges[adj[u][i]].v);

}

}

return lvl[t] != -1;

}

int dfs(int u, long long pushed) {

if(!pushed || u == t)

return pushed;

for(int& i = ptr[u]; i < adj[u].size(); i++) {

int idx = adj[u][i];

int v = edges[idx].v;

if(lvl[v] != lvl[u] + 1)

continue;

long long x = dfs(v, min(pushed, edges[idx].capacity - edges[idx].flow));

if(x > 0) {

edges[idx ^ 0].flow += x;

edges[idx ^ 1].flow -= x;

return x;

}

}

return 0;

}

long long maxFlow() {

long long flow = 0, x;

while(bfs())

while(x = dfs(s, INF))

flow += x;

return flow;

}

};

**L-R Flow using Dinic**

//flow\_through[i] = extra flow beyond 'low' sent through edge i

struct LR\_Flow {

Dinic F;

int n, s, t;

struct edge {

int u, v, l, r, id;

};

vector<edge> edges;

LR\_Flow() {}

LR\_Flow(int \_n) {

n = \_n + 10;

s = n - 2, t = n - 1;;

edges.clear();

}

void add\_edge(int u, int v, int l, int r, int id = -1) {

assert(0 <= l && l <= r);

edges.push\_back({u, v, l, r, id});

}

bool feasible(int \_s = -1, int \_t = -1, int L = -1, int R = -1) {

if (L != -1) edges.push\_back({\_t, \_s, L, R, -1});

F = Dinic(n);

long long target = 0;

for (auto e : edges) {

int u = e.u, v = e.v, l = e.l, r = e.r, id = e.id;

if (l != 0) {

F.add\_edge(s, v, l);

F.add\_edge(u, t, l);

target += l;

}

F.add\_edge(u, v, r - l, id);

}

auto ans = F.max\_flow(s, t);

if (L != -1) edges.pop\_back();

if (ans < target) return 0; //not feasible

return 1;

}

int max\_flow(int \_s, int \_t) { //-1 means flow is not feasible

int mx = 1e5 + 9;

if (!feasible(\_s, \_t, 0, mx)) return -1;

return F.max\_flow(\_s, \_t);

}

int min\_flow(int \_s, int \_t) { //-1 means flow is not feasible

int mx = 1e9;

int ans = -1, l = 0, r = mx;

while (l <= r) {

int mid = l + r >> 1;

if (feasible(\_s, \_t, 0, mid)) ans = mid, r = mid - 1;

else l = mid + 1;

}

return ans;

**MCMF O(F\*N\*M)**

const int mxN = 110;

const int inf = 2e9;

struct Edgee {

int to, cost, cap, flow, backEdge;

};

struct MCMF {

int s, t, n;

vector<Edgee> g[mxN];

MCMF(int \_s, int \_t, int \_n) {

s = \_s, t = \_t, n = \_n;

}

void addEdge(int u, int v, int cost, int cap) {

Edgee e1 = { v, cost, cap, 0, g[v].size() };

Edgee e2 = { u, -cost, 0, 0, g[u].size() };

g[u].push\_back(e1);

g[v].push\_back(e2);

}

pair<int, int> minCostMaxFlow() {

int flow = 0, cost = 0;

vector<int> state(n), from(n), from\_edge(n), d(n);

deque<int> q;

while (true) {

for (int i = 0; i < n; i++)

state[i] = 2, d[i] = inf, from[i] = -1;

state[s] = 1;

q.clear();

q.push\_back(s);

d[s] = 0;

while (!q.empty()) {

int v = q.front();

q.pop\_front();

state[v] = 0;

for (int i = 0; i < (int) g[v].size(); i++) {

Edgee e = g[v][i];

if (e.flow >= e.cap || d[e.to] <= d[v] + e.cost)

continue;

int to = e.to;

d[to] = d[v] + e.cost;

from[to] = v;

from\_edge[to] = i;

if (state[to] == 1) continue;

if (!state[to] || (!q.empty() && d[q.front()] > d[to]))

q.push\_front(to);

else q.push\_back(to);

state[to] = 1;

}

}

if (d[t] == inf) break;

int it = t, addflow = inf;

while (it != s) {

addflow = min(addflow,

g[from[it]][from\_edge[it]].cap

- g[from[it]][from\_edge[it]].flow);

it = from[it];

}

it = t;

while (it != s) {

g[from[it]][from\_edge[it]].flow += addflow;

g[it][g[from[it]][from\_edge[it]].backEdge].flow -= addflow;

cost += g[from[it]][from\_edge[it]].cost \* addflow;

it = from[it];

}

flow += addflow;

}

return {cost,flow};

}

};

**BPM O(sqrt(n)\*m)**

struct bpm {

const int inf = 0x3f3f3f3f;

vector <vector <int> > G;

vector <int> match, dist;

int n, m;

bpm(int n, int m) : n(n), m(m) {

G.resize(n + 1);

match.resize(n + m + 1, 0);

dist.resize(n + 1);

}

void add\_edge(int i, int j) {

G[i].push\_back(n + j);

}

bool bfs() {// here SPSF used .. Dijstr cn use

queue <int> Q;

fill(dist.begin(), dist.end(), inf);

for(int i = 1; i <= n; i++) if(!match[i]) {

dist[i] = 0;

Q.push(i);

}

while(Q.size()) {

int u = Q.front();

Q.pop();

if(!u) continue;

for(int v: G[u]) if(dist[match[v]] == inf) {

dist[match[v]] = 1 + dist[u];

Q.push(match[v]);

}

}

return dist[0] != inf;

}

bool dfs(int u) {

if(!u) return true;

for(int x: G[u]) {

int v = match[x];

if(dist[v] == dist[u] + 1 && dfs(v)) {

match[u] = x;

match[x] = u;

return true;

}

}

dist[u] = inf;

return false;

}

int get() {

int ans = 0;

while(bfs()) {

for(int i = 1; i <= n; i++) if(!match[i] && dfs(i)) ++ans;

}

return ans;

}

};

**Hungarian O(n\*n\*n)**

const int N = 509;

/\* Complexity: O(n^3) but optimized

It finds minimum cost maximum matching.

For finding maximum cost maximum matching

add -cost and return -matching()

1-indexed \*/

struct Hungarian {

long long c[N][N], fx[N], fy[N], d[N];

int l[N], r[N], arg[N], trace[N];

queue<int> q;

int start, finish, n;

const long long inf = 1e18;

Hungarian() {}

Hungarian(int n1, int n2): n(max(n1, n2)) {

for (int i = 1; i <= n; ++i) {

fy[i] = l[i] = r[i] = 0;

for (int j = 1; j <= n; ++j) c[i][j] = inf; // make it 0 for maximum cost matching (not necessarily with max count of matching)

}

}

void add\_edge(int u, int v, long long cost) {

c[u][v] = min(c[u][v], cost);

}

inline long long getC(int u, int v) {

return c[u][v] - fx[u] - fy[v];

}

void initBFS() {

while (!q.empty()) q.pop();

q.push(start);

for (int i = 0; i <= n; ++i) trace[i] = 0;

for (int v = 1; v <= n; ++v) {

d[v] = getC(start, v);

arg[v] = start;

}

finish = 0;

}

void findAugPath() {

while (!q.empty()) {

int u = q.front();

q.pop();

for (int v = 1; v <= n; ++v) if (!trace[v]) {

long long w = getC(u, v);

if (!w) {

trace[v] = u;

if (!r[v]) {

finish = v;

return;

}

q.push(r[v]);

}

if (d[v] > w) {

d[v] = w;

arg[v] = u;

}

}

}

}

void subX\_addY() {

long long delta = inf;

for (int v = 1; v <= n; ++v) if (trace[v] == 0 && d[v] < delta) {

delta = d[v];

}

// Rotate

fx[start] += delta;

for (int v = 1; v <= n; ++v) if(trace[v]) {

int u = r[v];

fy[v] -= delta;

fx[u] += delta;

} else d[v] -= delta;

for (int v = 1; v <= n; ++v) if (!trace[v] && !d[v]) {

trace[v] = arg[v];

if (!r[v]) {

finish = v;

return;

}

q.push(r[v]);

}

}

void Enlarge() {

do {

int u = trace[finish];

int nxt = l[u];

l[u] = finish;

r[finish] = u;

finish = nxt;

} while (finish);

}

long long maximum\_matching() {

for (int u = 1; u <= n; ++u) {

fx[u] = c[u][1];

for (int v = 1; v <= n; ++v) {

fx[u] = min(fx[u], c[u][v]);

}

}

for (int v = 1; v <= n; ++v) {

fy[v] = c[1][v] - fx[1];

for (int u = 1; u <= n; ++u) {

fy[v] = min(fy[v], c[u][v] - fx[u]);

}

}

for (int u = 1; u <= n; ++u) {

start = u;

initBFS();

while (!finish) {

findAugPath();

if (!finish) subX\_addY();

}

Enlarge();

}

long long ans = 0;

for (int i = 1; i <= n; ++i) {

if (c[i][l[i]] != inf) ans += c[i][l[i]];

else l[i] = 0;

}

return -ans;

}

};

**Number Theory**: **CRT**

struct CRT {

#define ll long long

vector<ll> v, p;

ll x, y, gcd, tempx, tempy,n;

CRT(vector<ll> \_v, vector<ll> \_p): v(\_v), p(\_p) {

n = \_v.size();

}

ll ex\_GCD(ll b, ll c, ll &x, ll &y) { // (1/b)%c = x

if(c==0) {

x = 1, y = 0;

return b;

}

gcd = ex\_GCD(c, b%c, x, y);

tempx = x, tempy = y;

x = tempy;

y = tempx - b/c\*tempy;

return gcd;

}

ll get() {

ll res = v[0], lcm = p[0];

for(ll i=1; i<n; i++) {

gcd = ex\_GCD(lcm, p[i], x, y);

if((v[i]-res)%gcd) {

return -1;

}

res = (res + x\*(v[i]-res)/gcd%(p[i]/gcd)\*lcm)%(lcm\*p[i]/gcd);

if(res < 0) res+=(lcm\*p[i]/gcd);

lcm = lcm \* p[i]/gcd;

}

return res;

}

#undef ll

};

**MISC**

#define N 1000000

#define lg 25

int d[lg\*N],nxt[lg\*N],lst[N+2],phi[N+2];

void eulerTotient() {

for(int i=2; i<=N; i++)phi[i]= i;

for(int i=2; i<=N; i++)

if(phi[i]==i) {

phi[i]--;

for(int j=2\*i; j<=N; j+=i)

phi[j]/= i, phi[j]\*= (i-1);

}

return;

}

long long phiSum[N+2];

void eulerTotientSum() {

for(int i=2; i<=N; i++)lst[i]= i;

for(int i=2, idx= N; i<=N; i++)

for(int j=i; j<=N; j+=i) {

idx++;

d[idx]= i;

nxt[ lst[j] ]= idx;

nxt[idx]= -1;

lst[j]= idx;

}

phiSum[1]= 1;

for(int j=2; j<=N; j++) {

phiSum[j]= (j\*1ll\*(j+1))/2;

int now= nxt[j];

while(now!=-1) {

int x= d[now];

phiSum[j]-= phiSum[j/x]\*1ll\*x;

now= nxt[now];

}

}

return;

}

namespace nt {

mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

int64\_t rand(int64\_t l, int64\_t r) {

return l + rng() % (r - l + 1);

}

int64\_t inverse(int64\_t a, int64\_t m) {

int64\_t x, y;

int64\_t g = gcd(a, m, x, y);

if(g > 1) return -1;

x = (x % m + m) % m;

return x;

}

// Returns minimum x for which a ^ x % m = b % m, a and m are coprime.

int solve(int a, int b, int m) {

a %= m, b %= m;

/\*int k = 1, add = 0, g;

while ((g = gcd(a, m)) > 1) {

if (b == k)

return add; k\* a ^ x % m = b % m

if (b % g)

return -1;

b /= g, m /= g, ++add;

k = (k \* 1ll \* a / g) % m;

}\*/

int n = sqrt(m) + 1, an = 1;

for (int i = 0; i < n; ++i) an = (an \* 1LL \* a) % m;

unordered\_map<int, int> vals;

for (int q = 0, cur = b; q <= n; ++q) {

vals[cur] = q, cur = (cur \* 1LL \* a) % m;

}

for (int p = 1, cur = 1; p <= n; ++p) {

cur = (cur \* 1LL \* an) % m;

if (vals.count(cur)) {

int ans = n \* p - vals[cur];

return ans;

}

}

return -1;

}

**Finding the number of solution**

void shift\_solution(int & x, int & y, int a, int b, int cnt) {

x += cnt \* b;

y -= cnt \* a;

}

int find\_all\_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {

int x, y, g;

if (!find\_any\_solution(a, b, c, x, y, g))

return 0;

a /= g;

b /= g;

int sign\_a = a > 0 ? +1 : -1;

int sign\_b = b > 0 ? +1 : -1;

shift\_solution(x, y, a, b, (minx - x) / b);

if (x < minx)

shift\_solution(x, y, a, b, sign\_b);

if (x > maxx)

return 0;

int lx1 = x;

shift\_solution(x, y, a, b, (maxx - x) / b);

if (x > maxx)

shift\_solution(x, y, a, b, -sign\_b);

int rx1 = x;

shift\_solution(x, y, a, b, -(miny - y) / a);

if (y < miny)

shift\_solution(x, y, a, b, -sign\_a);

if (y > maxy)

return 0;

int lx2 = x;

shift\_solution(x, y, a, b, -(maxy - y) / a);

if (y > maxy)

shift\_solution(x, y, a, b, sign\_a);

int rx2 = x;

if (lx2 > rx2)

swap(lx2, rx2);

int lx = max(lx1, lx2);

int rx = min(rx1, rx2);

if (lx > rx)

return 0;

return (rx - lx) / abs(b) + 1;

}

**SOD & NOD:**

n= p1^a1 \* p2^a2 \* p3^a3 \* ….. \* pk^ak

NOD: (a1+1)\*(a2+1)\*(a3+1)\*..\*(ak+1)

SOD: (p1^(a1+1)-1)/(p1-1) \* (p2^(a2+1)-1)/(p2-1) \*....\*(pk^(ak+1)-1)/(pk-1)

**N!%P:**

int factmod(int n, int p) {

vector<int> f(p);

f[0] = 1; for (int i = 1; i < p; i++)

f[i] = f[i-1] \* i % p;

int res = 1;

while (n > 1) {

if ((n/p) % 2)

res = p - res;

res = res \* f[n%p] % p;

n /= p;

}

return res;}

**Again Number theory**

// returns primes upto n

vector <int> sieve(int n) {

vector <int> prime, mn(n + 1, 0);

for(int i = 2; i <= n; i++) {

if(!mn[i]) {

prime.push\_back(i);

}

for(int j = 0; j < prime.size() && i \* prime[j] <= n; j++) {

mn[i \* prime[j]] = prime[j];

if(i % prime[j] == 0) break;

}

}

return prime;

}

namespace MillerRabin {

bool isComposite(int64\_t x, int64\_t a, int64\_t d, int64\_t s) {

int64\_t y = pow(a, d, x);

if(y == 1 || y == x - 1) return false;

for(int64\_t r = 1; r < s; r++) {

y = ( int128\_t)y \* y % x;

if(y == x - 1) return false;

}

return true;

}

bool isPrime(int64\_t x) {

if(x < 2) return false;

for(int64\_t a: sieve(40)) if(a == x) {

return true;

}

int64\_t r = 0, d = x - 1;

while(d % 2 == 0) {

d /= 2;

++r;

}

for(int64\_t a: sieve(40)) if(isComposite(x, a, d, r)) {

return false;

}

return true;

}

}

namespace Rho {

int64\_t f(int64\_t x, int64\_t c, int64\_t mod) {

return (( int128\_t)x \* x + c) % mod;

} int64\_t brent(int64\_t n) {

int64\_t x = rand(2, n), g = 1, q = 1, xs, y, c = rand(1, n);

int m = 128, l = 1;

while (g == 1) {

y = x;

for (int i = 1; i < l; i++)x = f(x, c, n);

int k = 0;

while (k < l && g == 1) {

xs = x;

for (int i = 0; i < m && i < l - k; i++) {

x = f(x, c, n);

q = ( int128\_t)q \* abs(y - x) % n;

}

g = gcd(q, n), k += m;

}

l \*= 2;

}

if (g == n) {

do {

xs = f(xs, c, n);

g = gcd(abs(xs - y), n);

} while (g == 1);

}

return g;

}

vector <int64\_t> factor(int64\_t n) {

if(n == 1) return {};

if(MillerRabin::isPrime(n)) {

return {n};

}

int64\_t dx = n;

while(dx == n) dx = brent(n);

auto L = factor(dx), R = factor(n / dx);

L.insert(L.end(), R.begin(), R.end());

return L;

}

}

**Math:** **FFT**

const int N = 2e5 + 9;

const double PI = acos(-1);

struct base {

double a, b;

base(double a = 0, double b = 0) : a(a), b(b) {}

const base operator + (const base &c) const {

return base(a + c.a, b + c.b);

}

const base operator - (const base &c) const {

return base(a - c.a, b - c.b);

}

const base operator \* (const base &c) const {

return base(a \* c.a - b \* c.b, a \* c.b + b \* c.a);

}

};

void fft(vector<base> &p, bool inv = 0) {

int n = p.size(), i = 0;

for(int j = 1; j < n - 1; ++j) {

for(int k = n >> 1; k > (i ^= k); k >>= 1);

if(j < i) swap(p[i], p[j]);

}

for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {

double ang = 2 \* PI / m;

base wn = base(cos(ang), (inv ? 1. : -1.) \* sin(ang)), w;

for(int i = 0, j, k; i < n; i += m) {

for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w \* wn) {

base t = w \* p[j + l];

p[j + l] = p[j] - t;

p[j] = p[j] + t;

}

}

}

if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /= n;

}

vector<long long> multiply(vector<int> &a, vector<int> &b) {

int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;

while(sz < t) sz <<= 1;

vector<base> x(sz), y(sz), z(sz);

for(int i = 0 ; i < sz; ++i) {

x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);

y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);

}

fft(x), fft(y);

for(int i = 0; i < sz; ++i) z[i] = x[i] \* y[i];

fft(z, 1);

vector<long long> ret(sz);

for(int i = 0; i < sz; ++i) ret[i] = (long long) round(z[i].a);

while((int)ret.size() > 1 && ret.back() == 0) ret.pop\_back();

return ret;

}

// ans[k] = sum(a[i] \* b[j]) over 0 <= i < n and j - i = k

vector<long long> cyclic\_convolution(vector<int> a, vector<int> b) {

assert(a.size() == b.size());

int n = a.size();

b.resize(3 \* n);

a.resize(3 \* n);

for (int i = 0; i < n; i++) {

b[i + n] = b[i + 2 \* n] = b[i];

a[i + n] = a[i + 2 \* n] = a[i];

}

auto c = multiply(a, b);

vector<long long> ans(n, 0);

for (int i = 0; i < n; i++) {

int j = i + n - 1;

ans[i] = c[i + j];

if (i) ans[i] -= c[2 \* i - 1];

}

return ans;

}

**FFT- any** **MOD**

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

struct base {

double x, y;

base() {

x = y = 0;

}

base(double x, double y): x(x), y(y) { }

};

inline base operator + (base a, base b) {

return base(a.x + b.x, a.y + b.y);

}

inline base operator - (base a, base b) {

return base(a.x - b.x, a.y - b.y);

}

inline base operator \* (base a, base b) {

return base(a.x \* b.x - a.y \* b.y, a.x \* b.y + a.y \* b.x);

}

inline base conj(base a) {

return base(a.x, -a.y);

}

int lim = 1;

vector<base> roots = {{0, 0}, {1, 0}};

vector<int> rev = {0, 1};

const double PI = acosl(- 1.0);

void ensure\_base(int p) {

if(p <= lim) return;

rev.resize(1 << p);

for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (p - 1));

roots.resize(1 << p);

while(lim < p) {

double angle = 2 \* PI / (1 << (lim + 1));

for(int i = 1 << (lim - 1); i < (1 << lim); i++) {

roots[i << 1] = roots[i];

double angle\_i = angle \* (2 \* i + 1 - (1 << lim));

roots[(i << 1) + 1] = base(cos(angle\_i), sin(angle\_i));

}

lim++;

}

}

void fft(vector<base> &a, int n = -1) {

if(n == -1) n = a.size();

assert((n & (n - 1)) == 0);

int zeros = \_\_builtin\_ctz(n);

ensure\_base(zeros);

int shift = lim - zeros;

for(int i = 0; i < n; i++) if(i < (rev[i] >> shift)) swap(a[i], a[rev[i] >> shift]);

for(int k = 1; k < n; k <<= 1) {

for(int i = 0; i < n; i += 2 \* k) {

for(int j = 0; j < k; j++) {

base z = a[i + j + k] \* roots[j + k];

a[i + j + k] = a[i + j] - z;

a[i + j] = a[i + j] + z;

}

}

}

}

//eq = 0: 4 FFTs in total

//eq = 1: 3 FFTs in total

vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {

int need = a.size() + b.size() - 1;

int p = 0;

while((1 << p) < need) p++;

ensure\_base(p);

int sz = 1 << p;

vector<base> A, B;

if(sz > (int)A.size()) A.resize(sz);

for(int i = 0; i < (int)a.size(); i++) {

int x = (a[i] % mod + mod) % mod;

A[i] = base(x & ((1 << 15) - 1), x >> 15);

}

fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});

fft(A, sz);

if(sz > (int)B.size()) B.resize(sz);

if(eq) copy(A.begin(), A.begin() + sz, B.begin());

else {

for(int i = 0; i < (int)b.size(); i++) {

int x = (b[i] % mod + mod) % mod;

B[i] = base(x & ((1 << 15) - 1), x >> 15);

}

fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});

fft(B, sz);

}

double ratio = 0.25 / sz;

base r2(0, - 1), r3(ratio, 0), r4(0, - ratio), r5(0, 1);

for(int i = 0; i <= (sz >> 1); i++) {

int j = (sz - i) & (sz - 1);

base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) \* r2;

base b1 = (B[i] + conj(B[j])) \* r3, b2 = (B[i] - conj(B[j])) \* r4;

if(i != j) {

base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) \* r2;

base d1 = (B[j] + conj(B[i])) \* r3, d2 = (B[j] - conj(B[i])) \* r4;

A[i] = c1 \* d1 + c2 \* d2 \* r5;

B[i] = c1 \* d2 + c2 \* d1;

}

A[j] = a1 \* b1 + a2 \* b2 \* r5;

B[j] = a1 \* b2 + a2 \* b1;

}

fft(A, sz);

fft(B, sz);

vector<int> res(need);

for(int i = 0; i < need; i++) {

long long aa = A[i].x + 0.5;

long long bb = B[i].x + 0.5;

long long cc = A[i].y + 0.5;

res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;}

return res;

}

**FWHT**

const int inv2 = (mod + 1) >> 1;

#define M (1 << 20)

#define OR 0

#define AND 1

#define XOR 2

struct FWHT {

int P1[M], P2[M];

void wt(int \*a, int n, int flag = XOR) {

if (n == 0) return;

int m = n / 2;

wt(a, m, flag);

wt(a + m, m, flag);

for (int i = 0; i < m; i++) {

int x = a[i], y = a[i + m];

if (flag == OR) a[i] = x, a[i + m] = (x + y) % mod;

if (flag == AND) a[i] = (x + y) % mod, a[i + m] = y;

if (flag == XOR) a[i] = (x + y) % mod, a[i + m] = (x - y + mod) % mod;

}

}

void iwt(int\* a, int n, int flag = XOR) {

if (n == 0) return;

int m = n / 2;

iwt(a, m, flag);

iwt(a + m, m, flag);

for (int i = 0; i < m; i++) {

int x = a[i], y = a[i + m];

if (flag == OR) a[i] = x, a[i + m] = (y - x + mod) % mod;

if (flag == AND) a[i] = (x - y + mod) % mod, a[i + m] = y;

if (flag == XOR) a[i] = 1LL \* (x + y) \* inv2 % mod, a[i + m] = 1LL \* (x - y + mod) \* inv2 % mod;

// replace inv2 by >>1 if not required

}

}

vector<int> multiply(int n, vector<int> A, vector<int> B, int flag = XOR) {

assert(\_\_builtin\_popcount(n) == 1);

A.resize(n);

B.resize(n);

for (int i = 0; i < n; i++) P1[i] = A[i];

for (int i = 0; i < n; i++) P2[i] = B[i];

wt(P1, n, flag);

wt(P2, n, flag);

for (int i = 0; i < n; i++) P1[i] = 1LL \* P1[i] \* P2[i] % mod;

iwt(P1, n, flag);

return vector<int> (P1, P1 + n);

}

vector<int> pow(int n, vector<int> A, long long k, int flag = XOR) {

assert(\_\_builtin\_popcount(n) == 1);

A.resize(n);

for (int i = 0; i < n; i++) P1[i] = A[i];

wt(P1, n, flag);

for(int i = 0; i < n; i++) P1[i] = POW(P1[i], k);

iwt(P1, n, flag);

return vector<int> (P1, P1 + n);

}

} t;

**String**: **Z-Algo**

vector<int> z\_function(string s) {

int n = (int) 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;

}

**Manacher**

vector<int> manacher\_odd(string s) {

int n = s.size();

s = "$" + s + "^";

vector<int> p(n + 2);

int l = 1, r = 1;

for(int i = 1; i <= n; i++) {

p[i] = max(0, min(r - i, p[l + (r - i)]));

while(s[i - p[i]] == s[i + p[i]]) {

p[i]++;

}

if(i + p[i] > r) {

l = i - p[i], r = i + p[i];

}

}

return vector<int>(begin(p) + 1, end(p) - 1);

}

//working with parities

vector<int> manacher(string s) {

string t;

for(auto c: s) {

t += string("#") + c;

}

auto res = manacher\_odd(t + "#");

return vector<int>(begin(res) + 1, end(res) - 1)

}

**Aho-corasick**

struct AC {

static const int K = 26;

struct node {

int nxt[K], link, leaf, par;

char pch;

node(int par = -1, char pch = '$') : par(par), pch(pch) {

fill(begin(nxt), end(nxt), -1);

leaf = false;

link = -1;

}

};

vector <node> t;

vector <vector <int> > ad;

vector <int> st, en, mp;

int dt = 0;

AC() {

mp = vector <int> (1 << 8, 0);

for(char ch = 'a'; ch <= 'z'; ++ch) mp[ch] = ch - 'a';

t.resize(1);

}

int add(string str) {

int ptr = 0;

for(auto ch: str) {

if(t[ptr].nxt[mp[ch]] < 0) {

t[ptr].nxt[mp[ch]] = t.size();

t.push\_back(node(ptr, ch));

}

ptr = t[ptr].nxt[mp[ch]];

}

t[ptr].leaf = true;

return ptr;

}

int get\_link(int v) {

if(t[v].link == -1) {

t[v].link = (!v || !t[v].par) ? 0 : go(get\_link(t[v].par), t[v].pch);

}

return t[v].link;

}

int go(int v, char ch) {

if(t[v].nxt[mp[ch]] < 0) {

t[v].nxt[mp[ch]] = !v ? 0 : go(get\_link(v), ch);

}

return t[v].nxt[mp[ch]];

}

void dfs(int v) {

st[v] = ++dt;

for(int u: ad[v])

dfs(u);

en[v] = dt;

}

void calc() {

int k = t.size();

st.resize(k);

en.resize(k);

ad.resize(k);

for(int i = 1; i < k; i++) ad[get\_link(i)].push\_back(i);

dfs(0);

}

} ac;

**Suffix-Automata**

struct SA {

struct state {

int len, link, nxt[26];

state() : len(0), link(-1) {

fill(begin(nxt), end(nxt), 0);

}

};

vector <state> d;

int last = 0;

SA() {

d.push\_back(state());

}

void add(char ch) {

int cur = d.size();

d.push\_back(state());

d[cur].len = d[last].len + 1;

int p = last;

while(p != -1 && !d[p].nxt[ch - 'a']) {

d[p].nxt[ch - 'a'] = cur;

p = d[p].link;

}

if(p == -1) {

d[cur].link = 0;

} else {

int q = d[p].nxt[ch - 'a'];

if(d[q].len == d[p].len + 1) {

d[cur].link = q;

} else {

int clone = d.size();

d.push\_back(d[q]);

d[clone].len = d[p].len + 1;

while(p != -1 && d[p].nxt[ch - 'a'] == q) {

d[p].nxt[ch - 'a'] = clone;

p = d[p].link;

}

d[cur].link = d[q].link = clone;

}

}

last = cur;

}

};

**Suffix-Array**

struct SA {

int n;

vector <int> lcp, sa, rank;

vector <vector <int> > t;

SA() {}

SA(string str) : n(str.size()) {

vector <int> p(n), c(n), cnt(max(1 << 8, n), 0);

for(int i = 0; i < n; i++) cnt[str[i]]++;

for(int i = 1; i < (1 << 8); i++) cnt[i] += cnt[i - 1];

for(int i = 0; i < n; i++) p[--cnt[str[i]]] = i;

int cc = 1;

c[p[0]] = 0;

for(int i = 1; i < n; i++) {

cc += str[p[i]] != str[p[i - 1]];

c[p[i]] = cc - 1;

}

vector <int> pn(n), cn(n);

for(int h = 0; (1 << h) < n; h++) {

for(int i = 0; i < n; i++) pn[i] = (p[i] - (1 << h) + n) % n;

fill(cnt.begin(), cnt.begin() + cc, 0);

for(int i = 0; i < n; i++) cnt[c[pn[i]]]++;

for(int i = 1; i < cc; i++) cnt[i] += cnt[i - 1];

for(int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];

cc = 1;

cn[p[0]] = 0;

for(int i = 1; i < n; i++) {

pair <int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]}, prv = {c[p[i - 1]], c[(p[i - 1] + (1 << h)) % n]};

cc += (prv != cur);

cn[p[i]] = cc - 1;

}

c.swap(cn);

}

sa = p;

rank.resize(n);

lcp.resize(n, 0);

for(int i = 0; i < n; i++) rank[sa[i]] = i;

int k = 0;

for(int i = 0; i < n; i++) {

if(rank[i] == n - 1) {

k = 0;

continue;

}

int j = p[rank[i] + 1];

while(i + k < n && j + k < n && str[i + k] == str[j + k]) ++k;

lcp[rank[i]] = k;

if(k) --k;

}

}

void build\_rmq() {

int l = 32 - \_\_builtin\_clz(n);

t = vector <vector <int> > (l, vector <int> (n, 0));

for(int i = 0; i < n; i++) t[0][i] = lcp[i];

for(int i = 1; i < l; i++) {

for(int j = 0; j + (1 << i) - 1 < n; j++) {

t[i][j] = min(t[i - 1][j], t[i - 1][j + (1 << (i - 1))]);

}

}

}

int query(int l, int r) {

int h = 31 - \_\_builtin\_clz(r - l + 1);

return min(t[h][l], t[h][r - (1 << h) + 1]);

}

int find\_left(int i, int k) {

int l = 1, r = rank[i] - 1, j = rank[i];

while(l <= r) {

int mid = (l + r) >> 1;

if(query(mid, rank[i] - 1) >= k) j = mid, r = mid - 1;

else l = mid + 1;

}

return j;

}

int find\_right(int i, int k) {

int l = rank[i] + 1, r = n - 1, j = rank[i];

while(l <= r) {

int mid = (l + r) >> 1;

if(query(rank[i], mid - 1) >= k) j = mid, l = mid + 1;

else r = mid - 1;

}

return j;

}

} sa;

**EERTREE**

struct eertree {

struct node {

int nxt[26], len, link;

node() : len(0), link(-1) {

fill(nxt, nxt + 26, 0);

}

};

vector <node> t;

int p;

eertree() : p(2) {

t = vector <node> (3);

t[1].len = -1;

t[2].len = 0;

t[2].link = t[1].link = 1;

}

int add(int pos, string &str) {

while(str[pos - t[p].len - 1] != str[pos]) p = t[p].link;

int ch = str[pos] - 'a', x = t[p].link, r = 0;

while(str[pos - t[x].len - 1] != str[pos]) x = t[x].link;

if(!t[p].nxt[ch]) {

r = 1;

int y = t.size();

t[p].nxt[ch] = y;

t.push\_back(node());

t[y].len = t[p].len + 2;

t[y].link = t[y].len == 1 ? 2 : t[x].nxt[ch];

}

p = t[p].nxt[ch];

return r;

}

};

**Double Hash**

const int mod1=1000001011,mod2=1000009999;

const int b1=193,b2=307;//greater than maximum value

//LL mod[5] = {1000000000000000003, 1000000000000037, //1000000000039, 1000000007, 1000003}, base[5] = {211, 307, //401, 503, 163};

int pw1[N+2],pw2[N+2];

void calPower() {

pw1[0]=1;

pw2[0]=1;

for(int i=1; i<=N; i++) {

pw1[i]=(1LL\*pw1[i-1]\*b1)%mod1;

pw2[i]=(1LL\*pw2[i-1]\*b2)%mod2;

}

}

struct hsh {

vector<pair<int,int> >v;

string s;

void init(string \_s) {

v.clear();

s=\_s;

}

pair<int,int>makeHash() {

int hs1=0,hs2=0;

for(auto x:s) {

hs1=(1LL\*hs1\*b1+x)%mod1;

hs2=(1LL\*hs2\*b2+x)%mod2;

v.push\_back({hs1,hs2});

}

return {hs1,hs2};

}

pair<int,int>segment(int lt,int rt) {

int x=v[rt].first,y=v[rt].second;

if(!lt)return {x,y};

x=x-(1LL\*v[lt-1].first\*pw1[rt-lt+1])%mod1;

y=y-(1LL\*v[lt-1].second\*pw2[rt-lt+1])%mod2;

if(x<0)x+=mod1;

if(y<0)y+=mod2;

return {x,y};

}

} hs;

**Miscellaneous**: **Closest pair point**

const int N = 3e5 + 9;

#define x first

#define y second

long long dist2(pair<int, int> a, pair<int, int> b) {

return 1LL \* (a.x - b.x) \* (a.x - b.x) + 1LL \* (a.y - b.y) \* (a.y - b.y);}

pair<int, int> closest\_pair(vector<pair<int, int>> a) {

int n = a.size();

assert(n >= 2);

vector<pair<pair<int, int>, int>> p(n);

for (int i = 0; i < n; i++) p[i] = {a[i], i};

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

int l = 0, r = 2;

long long ans = dist2(p[0].x, p[1].x);

pair<int, int> ret = {p[0].y, p[1].y};

while (r < n) {

while (l < r && 1LL \* (p[r].x.x - p[l].x.x) \* (p[r].x.x - p[l].x.x) >= ans) l++;

for (int i = l; i < r; i++) {

long long nw = dist2(p[i].x, p[r].x);

if (nw < ans) {

ans = nw;

ret = {p[i].y, p[r].y};

}

}

r++;

}

return ret;

}

**Iterative Ex\_GECD**

int gcd(int a, int b, int& x, int& y) {

x = 1, y = 0;

int x1 = 0, y1 = 1, a1 = a, b1 = b;

while (b1) {

int q = a1 / b1;

tie(x, x1) = make\_tuple(x1, x - q \* x1);

tie(y, y1) = make\_tuple(y1, y - q \* y1);

tie(a1, b1) = make\_tuple(b1, a1 - q \* b1);

}

return a1;

}

**Starling Number:**

/\*stirling number of seocond kind:

there are n different object(distinguishable) and k identical boxes

(indistinguishable). how many ways to put object into boxes such that each box

have atleast one object.

boxes are considered to be set. permutation of object in box doesn't matter.

if boxes are distinguishable multiply by k!

1 k

formula: --- ∑ (-1)^i kCi (k-i)^n

k! i=0

both k can be crossed out. after reformulating:

(-1)^i 1/(i!) \* 1/(k-i)! (k-i)^n

for fixed n we can calculte for all k using fft

for stirling(3,2) there are three ways {1|23} {2|13} {3|12}\*/

int fac[N+2];

void init(int n)

{

fac[0]=1;for(int i=1;i<=n;i++)fac[i]=(1LL\*fac[i-1]\*i)%mod;

}

void stirling(int n,int k)

{

init(N);vector<int>a,b,c;

for(int i=0;i<=n;i++)

{

a.push\_back((big(-1,i)\*big(fac[i],mod-2)+mod)%mod);

b.push\_back((1LL\*big(i,n)\*big(fac[i],mod-2))%mod);

}

c=multiply(a,b);//ntt multiply

//each index in c means stirling(n,i)

}

**Berlekamp-Massey**

/\*complexity m^2 log(n) - m^3

m = length of recurrence\*/

const int mod=1e9+7;

int big(int b,int p) {

int ret=1;

while(p) {

if(p&1)ret=(1LL\*ret\*b)%mod;

b=(1LL\*b\*b)%mod;

p/=2;

}

return ret;

}

vector<int>berlekampMassey(vector<int>x) {

vector<int>ls,cur;

int lf,ld;

for(int i=0; i<(int)x.size(); i++) {

int t=0;

for(int j=0; j<(int)cur.size(); j++)t=(t+1LL\*x[i-j-1]\*cur[j])%mod;

if((t-x[i])%mod==0)continue;

if(!cur.size()) {

cur.resize(i+1);

lf=i;

ld=(t-x[i])%mod;

continue;

}

int k=-1LL\*(x[i]-t)\*big(ld,mod-2)%mod;

vector<int>c(i-lf-1);

c.push\_back(k);

for(int j=0; j<ls.size(); j++)c.push\_back((-1LL\*ls[j]\*k)%mod);

if((int)c.size()<(int)cur.size())c.resize((int)cur.size());

for(int j=0; j<(int)cur.size(); j++)

c[j]=(c[j]+cur[j])%mod;

if(i-lf+(int)ls.size()>=(int)cur.size())

ls=cur,lf=i,ld=(t-x[i])%mod;

cur=c;

}

for(int i=0; i<cur.size(); i++)

cur[i]=(cur[i]%mod+mod)%mod;

return cur;

}

int m;

int a[N+2],h[N+2],t\_[N+2],s[N+2],t[N+2];

void mul(int\*p,int\*q) {

for(int i=0; i<m+m; i++)t\_[i]=0;

for(int i=0; i<m; i++)if(p[i])

for(int j=0; j<m; ++j)

t\_[i+j]=(t\_[i+j]+1LL\*p[i]\*q[j])%mod;

for(int i=m+m-1; i>=m; i--)if(t\_[i])

for(int j=m-1; ~j; j--)

t\_[i-j-1]=(t\_[i-j-1]+1LL\*t\_[i]\*h[j])%mod;

for(int i=0; i<m; ++i)p[i]=t\_[i];

}

int cal(int k) {

for(int i=m; ~i; --i)s[i]=t[i]=0;

s[0]=1;

if(m!=1)t[1]=1;

else t[0]=h[0];

while(k) {

if(k&1)mul(s,t);

mul(t,t);

k>>=1;

}

int su=0;

for(int i=0; i<m; i++)su=(su+1LL\*s[i]\*a[i])%mod;

return (su%mod+mod)%mod;

}

void work(vector<int>x) { //first element should be non zero

vector<int>v=berlekampMassey(x);

m=v.size();

for(int i=0; i<m; i++)h[i]=v[i],a[i]=x[i];

}//index:0 1 2 3 4 5 6 7

//work({1,1,2,3,5,8,13,21});//cal(8)=34,cal(10)=89

**Link-Cut Tree**

/\*represented tree: the actual tree. tree we want to

represent. actual tree are forest of some rooted trees.

like dsu tree.\*

preferred child: u is called preferred child of v if

last accessed node in v's subtree is in u's subtree.

preferred child can be null if v is last accessed node

or v has no accessed node.\*

preferred edge: edge between node and preferred child.\*

preferred path: continuous path where each edge is

preferred edge. preferred path can be a single node.

a tree will have many preferred path. each preferred

path is node disjoint and every node is in exactly one

preferred path.\*

splay tree: splay trees represent

preferred path. one splay tree for each path. splay

trees are balanced binary search tree where nodes are

keyed by their depth in actual tree.\*

path-parent pointer: path parent pointer connects two

splay trees. path parent pointer is stored in the root

of splay tree and points to parent of the topmost

(lowest depth) node of preferred path. so a path parent

pointer is an edge of represented tree which is not

preferred.\*/

struct node {

int p,ch[2];//parent and child;

long dt,sm;//value of the node and sum of subtree(in splay)

bool rev;//splay tree operation.

int sz;//size of the subtree(in splay).

long lz;//lazy add

node() {}

node(int v):p(-1),dt(v),sm(v),rev(0),sz(1),lz(0) {

ch[0]=ch[1]=-1;

}

};

node t[N+2];

void tooLazy(int x) { //lazy propagation

if(t[x].lz) {

t[x].dt+=t[x].lz,t[x].sm+=t[x].lz\*t[x].sz;

if(t[x].ch[0]+1)t[t[x].ch[0]].lz+=t[x].lz;

if(t[x].ch[1]+1)t[t[x].ch[1]].lz+=t[x].lz;

}

if(t[x].rev) {

swap(t[x].ch[0],t[x].ch[1]);

if(t[x].ch[0]+1)t[t[x].ch[0]].rev^=1;

if(t[x].ch[1]+1)t[t[x].ch[1]].rev^=1;

}

t[x].lz=0,t[x].rev=0;

}

void cal(int x) { //splay tree operation

t[x].sz=1,t[x].sm=t[x].dt;

for(int i=0; i<2; i++) {

if(t[x].ch[i]+1==0)continue;

tooLazy(t[x].ch[i]);

t[x].sz+=t[t[x].ch[i]].sz;

t[x].sm+=t[t[x].ch[i]].sm;

}

}

bool isRoot(int x) {

return (t[x].p==-1)||((t[t[x].p].ch[0]!=x)&&(t[t[x].p].ch[1]!=x));

}

void rotate(int x) { //splay tree

int p=t[x].p,pp=t[p].p;

if(!isRoot(p))t[pp].ch[t[pp].ch[1]==p]=x;

bool d=t[p].ch[0]==x;

t[p].ch[!d]=t[x].ch[d],t[x].ch[d]=p;

if(t[p].ch[!d]+1)t[t[p].ch[!d]].p=p;

t[x].p=pp,t[p].p=x;

cal(p),cal(x);

}

int splay(int x) { //splay tree

while(!isRoot(x)) {

int p=t[x].p,pp=t[p].p;

if(!isRoot(p))tooLazy(pp);

tooLazy(p),tooLazy(x);

if(!isRoot(p))rotate((t[pp].ch[0]==p)^(t[p].ch[0]==x)?x:p);

rotate(x);

}

return tooLazy(x),x;

}

int access(int v) {

int last=-1;

for(int w=v; w+1; cal(last=w),splay(v),w=t[v].p)

splay(w),t[w].ch[1]=(last==-1?-1:v);

return last;

}

void init(int v,int w) {

t[v]=node(w); //node v initialization with value w.

}

int findRoot(int v) {

access(v),tooLazy(v);

while(t[v].ch[0]+1)v=t[v].ch[0],tooLazy(v);

return splay(v);

}

bool isConnected(int v,int w) {

access(v),access(w);

return v==w?true:t[v].p!=-1;

}

void makeRoot(int v) {

access(v);

t[v].rev^=1;

}

long query(int u,int v) { //query on path u-v

makeRoot(v),access(u);

return t[u].sm;

}

void update(int u,int v,int x) { //add x to all node in u-v path

makeRoot(v),access(u);

t[u].lz+=x;

}

void add(int u,int v) { //add an edge between u and v.

if(isConnected(u,v))return ;

makeRoot(v);

t[v].p=u;

}

void cut(int u,int v) { //cut edge between u and v, given they already have edge.

makeRoot(v),access(u);

t[u].ch[0]=t[t[u].ch[0]].p=-1;

}

int lca(int u,int v) { //lca between u and v.

access(u);

return access(v);

}

2**D-Geometry;**

#include <bits/stdc++.h>

#define inf 1e20

#pragma GCC optimize ("O2,unroll-loops")

using namespace std;

const double pi= 4\*atan(1), eps= 1e-14;

inline int dcmp (double x) { if (fabs(x) < eps) return 0; else return x < 0 ? -1 : 1; }

double fix\_acute(double th) {return th<-pi ? (th+2\*pi): th>pi ? (th-2\*pi) : th;}

inline double getDistance (double x, double y) { return sqrt(x \* x + y \* y); }

inline double torad(double deg) { return deg / 180 \* pi; }

struct Point {

double x, y;

Point (double x = 0, double y = 0): x(x), y(y) {}

void read () { scanf("%lf%lf", &x, &y); }

void write () { printf("%lf %lf", x, y); }

bool operator == (const Point& u) const { return dcmp(x - u.x) == 0 && dcmp(y - u.y) == 0; }

bool operator != (const Point& u) const { return !(\*this == u); }

bool operator < (const Point& u) const { return dcmp(x - u.x) < 0 || (dcmp(x-u.x)==0 && dcmp(y-u.y) < 0); }

bool operator > (const Point& u) const { return u < \*this; }

bool operator <= (const Point& u) const { return \*this < u || \*this == u; }

bool operator >= (const Point& u) const { return \*this > u || \*this == u; }

Point operator + (const Point& u) { return Point(x + u.x, y + u.y); }

Point operator - (const Point& u) { return Point(x - u.x, y - u.y); }

Point operator \* (const double u) { return Point(x \* u, y \* u); }

Point operator / (const double u) { return Point(x / u, y / u); }

double operator \* (const Point& u) { return x\*u.y - y\*u.x; }

};

typedef Point Vector;

typedef vector<Point> Polygon;

struct Line {

double a, b, c;

Line (double a = 0, double b = 0, double c = 0): a(a), b(b), c(c) {}

};

double toDeg(double rad){

return (rad\*180.0)/pi;

}

double modifiedatan2(Point a){

double ret= atan2(a.y, a.x);

if(ret<0)ret+= 2\*pi;

return toDeg(ret);

}

struct Segment{

Point a;

Point b;

Segment(){}

Segment(Point aa,Point bb) {a=aa,b=bb;}

};

struct DirLine {

Point p;

Vector v;

double ang;

DirLine () {}

DirLine (Point p, Vector v): p(p), v(v) { ang = atan2(v.y, v.x); }

bool operator < (const DirLine& u) const { return ang < u.ang; }

};

namespace Punctual {

double getDistance(Point a, Point b) { double x=a.x-b.x, y=a.y-b.y; return sqrt(x\*x + y\*y); }

};

namespace Vectorial {

bool cmpX(const Point &a, const Point &b){return a.x < b.x;}

bool cmpY(const Point &a, const Point &b){return a.y < b.y;}

bool cmpAngle(const Point &a, const Point &b){

/// sorts by angle (counter-clock wise by radian)

return modifiedatan2(a)<modifiedatan2(b);

}

double getDot (Vector a, Vector b) { return a.x \* b.x + a.y \* b.y; }

double getCross (Vector a, Vector b) { return a.x \* b.y - a.y \* b.x; }

double getLength (Vector a) { return sqrt(getDot(a, a)); }/// to get point a to point b distance, call getPLength(b-a)

double getPLength (Vector a) { return getDot(a, a); }

double getAngle (Vector u) { return atan2(u.y, u.x); }

double getSignedAngle (Vector a, Vector b) {return getAngle(b)-getAngle(a);}

Vector rotate (Vector a, double rad) { return Vector(a.x\*cos(rad)-a.y\*sin(rad), a.x\*sin(rad)+a.y\*cos(rad)); }

Vector ccw(Vector a, double co, double si) {return Vector(a.x\*co-a.y\*si, a.y\*co+a.x\*si);}

Vector cw (Vector a, double co, double si) {return Vector(a.x\*co+a.y\*si, a.y\*co-a.x\*si);}

Vector scale(Vector a, double s = 1.0) {return a / getLength(a) \* s;}

Vector getNormal (Vector a) { double l = getLength(a); return Vector(-a.y/l, a.x/l); }

};

namespace ComplexVector {

typedef complex<double> Point;

typedef Point Vector;

double getDot(Vector a, Vector b) { return real(conj(a)\*b); }

double getCross(Vector a, Vector b) { return imag(conj(a)\*b); }

Vector rotate(Vector a, double rad) { return a\*exp(Point(0, rad)); }

};

namespace Linear {

using namespace Punctual;

using namespace Vectorial;

Line getLine (double x1, double y1, double x2, double y2) { return Line(y2-y1, x1-x2, y1\*x2-x1\*y2); }

Line getLine (double a, double b, Point u) { return Line(a, -b, u.y \* b - u.x \* a); }

bool getIntersection (Line p, Line q, Point& o) {

if (fabs(p.a \* q.b - q.a \* p.b) < eps)

return false;

o.x = (q.c \* p.b - p.c \* q.b) / (p.a \* q.b - q.a \* p.b);

o.y = (q.c \* p.a - p.c \* q.a) / (p.b \* q.a - q.b \* p.a);

return true;

}

bool getIntersection (Point p, Vector v, Point q, Vector w, Point& o) {

if (dcmp(getCross(v, w)) == 0) return false;

Vector u = p - q;

double k = getCross(w, u) / getCross(v, w);

o = p + v \* k;

return true;

}

double closestPairPoint(Point\* P, int n){

typedef set<Point, bool(\*)(const Point&, const Point&)>setType;

typedef setType::iterator setIT;

setType s(&cmpY);

double ret= inf;

sort(P, P+n, cmpX);

for(int i=1; i<n; i++)

if(P[i-1].x==P[i].x && P[i-1].y==P[i].y)return 0.0;

s.clear();

for(int i=0; i<n; i++)s.insert(P[i]);

for(int i=0, idx=0; i<n; i++)

{

Point it= P[i];

while(it.x-P[idx].x>ret){

s.erase(P[idx]);idx++;

}

Point low= Point(it.x, it.y - ret);

Point high= Point(it.x, it.y + ret);

setIT lowest = s.lower\_bound(low);

if(lowest!=s.end())

{

setIT highest= s.upper\_bound(high);

for(setIT now= lowest; now!= highest; now++)

{

double cur= getDistance(\*now, it);

if(cur==0)continue;

ret= min(ret, cur);

}

}

s.insert(it);

}

return ret;

}

double perpendicularProjection(Point p, Point a, Point b){ /// from point p to line(a, b)

Point edge= (b-a)/getLength(b-a);

return getDot(edge, p-a);

}

double getDistanceToLine(Point p, Point a, Point b) { return fabs(getCross(b-a, p-a) / getLength(b-a)); }

double getDistanceToSegment(Point p, Point a, Point b) {

if (a == b) return getLength(p-a);

Vector v1 = b - a, v2 = p - a, v3 = p - b;

if (dcmp(getDot(v1, v2)) < 0) return getLength(v2);

else if (dcmp(getDot(v1, v3)) > 0) return getLength(v3);

else return fabs(getCross(v1, v2) / getLength(v1));

}

double getDistanceSegToSeg (Point a,Point b,Point c,Point d){

double Ans=INT\_MAX;

Ans=min(Ans,getDistanceToSegment(a,c,d));

Ans=min(Ans,getDistanceToSegment(b,c,d));

Ans=min(Ans,getDistanceToSegment(c,a,b));

Ans=min(Ans,getDistanceToSegment(d,a,b));

return Ans;

}

Point getPointToLine (Point p, Point a, Point b) { Vector v = b-a; return a+v\*(getDot(v, p-a) / getDot(v,v)); }

bool onSegment (Point p, Point a, Point b) { return dcmp(getCross(a-p, b-p)) == 0 && dcmp(getDot(a-p, b-p)) <= 0; }

bool haveIntersection (Point a1, Point a2, Point b1, Point b2) {

if(onSegment(a1,b1,b2)) return true;

if(onSegment(a2,b1,b2)) return true;

if(onSegment(b1,a1,a2)) return true;

if(onSegment(b2,a1,a2)) return true; ///Case of touching

double c1=getCross(a2-a1, b1-a1), c2=getCross(a2-a1, b2-a1), c3=getCross(b2-b1, a1-b1), c4=getCross(b2-b1,a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3)\*dcmp(c4) < 0;

}

bool onLeft(DirLine l, Point p) { return dcmp(l.v \* (p-l.p)) >= 0; }

};

namespace Triangular {

using namespace Vectorial;

double getAngle (double a, double b, double c) { return acos((a\*a+b\*b-c\*c) / (2\*a\*b)); }

double getArea (double a, double b, double c) { double s =(a+b+c)/2; return sqrt(s\*(s-a)\*(s-b)\*(s-c)); }

double getArea (double a, double h) { return a \* h / 2; }

double getArea (Point a, Point b, Point c) { return fabs(getCross(b - a, c - a)) / 2; }

double getDirArea (Point a, Point b, Point c) { return getCross(b - a, c - a) / 2;}

//ma/mb/mc = length of median from side a/b/c

double getArea\_(double ma,double mb,double mc) {double s=(ma+mb+mc)/2; return 4/3.0 \* sqrt(s\*(s-ma)\*(s-mb)\*(s-mc));}

//ha/hb/hc = length of perpendicular from side a/b/c

double get\_Area(double ha,double hb,double hc){

double H=(1/ha+1/hb+1/hc)/2; double \_A\_ = 4 \* sqrt(H \* (H-1/ha)\*(H-1/hb)\*(H-1/hc)); return 1.0/\_A\_;

}

bool pointInTriangle(Point a, Point b, Point c, Point p){

double s1 = getArea(a,b,c);

double s2 = getArea(p,b,c) + getArea(p,a,b) + getArea(p,c,a);

return dcmp(s1 - s2) == 0;

}

};

namespace Polygonal{

using namespace Vectorial;

using namespace Linear;

using namespace Triangular;

double getSignedArea (Point\* p, int n){

double ret = 0;

for (int i = 0; i < n-1; i++)

ret += (p[i]-p[0]) \* (p[i+1]-p[0]);

return ret/2.0;

}

long long pointsOnPolygon(Point\* p, int n){

long long ret= 0;

for(int i=0; i<n; i++){

Point a= p[(i+1)%n]-p[i];

long long g= abs(\_\_gcd((long long)a.x, (long long)a.y));

ret+= g;

}

return ret;

}

int getConvexHull (Point\* p, int n, Point\* ch){

sort(p, p + n);

/// preparing lower hull

int m = 0;

for (int i = 0; i < n; i++){

while (m > 1 && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-1])) <= 0) m--;

ch[m++] = p[i];

}

/// preparing upper hull

int k = m;

for (int i = n-2; i >= 0; i--){

while (m > k && dcmp(getCross(ch[m-1]-ch[m-2], p[i]-ch[m-2])) <= 0) m--;

ch[m++] = p[i];

}

if (n > 1) m--;

return m;

}

double diameter(Point\* p, int n, Point\* ch){

n= getConvexHull(p, n, ch);

double ret= 0;

for(int i=0, j=1; i<n; i++){

if(i==j)j= (j+1)%n;

while(getDistance(ch[i], ch[j])<getDistance(ch[i], ch[ (j+1)%n ]))j= (j+1)%n;

ret= max(ret, getDistance(ch[i], ch[j]));

}

return ret;

}

int isPointInPolygon(Point o, Point\* p, int n) {

int wn = 0;

for (int i = 0; i < n; i++) {

int j = (i + 1) % n;

if (onSegment(o, p[i], p[j]) || o == p[i]) return 0;

int k = dcmp(getCross(p[j] - p[i], o-p[i]));

int d1 = dcmp(p[i].y - o.y);

int d2 = dcmp(p[j].y - o.y);

if (k > 0 && d1 <= 0 && d2 > 0) wn++;

if (k < 0 && d2 <= 0 && d1 > 0) wn--;

}

return wn ? -1 : 1;

}

Polygon maximumEnclosingTriangle(Point\* p, int n)/// p is a array containing Points of convexHull

{

Polygon ret;

if(n<3)return ret;

double res= 0.0;

for(int i=0, j=1, k=2; i<n; i++){

if(i==j)j= (j+1)%n;

if(j==k)k= (k+1)%n;

double area= getArea(p[i], p[j], p[k]);

while(true)

{

while(true){

int nk= (k+1)%n;

double narea= getArea(p[i], p[j], p[nk]);

if(dcmp(narea-area)>=0)area= narea, k= nk;

else break;

}

int nj= (j+1)%n;

double narea= getArea(p[i], p[nj], p[k]);

if(dcmp(narea-area)>=0)area= narea, j= nj;

else break;

}

if(dcmp(area-res)>0)res= area, ret.clear(),

ret.push\_back(p[i]), ret.push\_back(p[j]), ret.push\_back(p[k]);

}

return ret;

}

pair<double, double>minimumEnclosingRectangle(Point \*p, int n, Point \*ch){

pair<double, double>ret= {1e9, 1e9};

n= getConvexHull(p, n, ch);

if(n<3)return ret;

for(int i=0; i<n; i++)p[i]= ch[i];

int l= 1, r= 1, u= 1;

for(int i=0; i<n; i++){

while(perpendicularProjection(p[(r+1)%n], p[i], p[(i+1)%n])>perpendicularProjection(p[r%n], p[i], p[(i+1)%n]))r++;

while(u<r || getDistanceToLine(p[(u+1)%n], p[i], p[(i+1)%n])>getDistanceToLine(p[u%n], p[i], p[(i+1)%n]))u++;

while(l<u || perpendicularProjection(p[(l+1)%n], p[i], p[(i+1)%n])<perpendicularProjection(p[l%n], p[i], p[(i+1)%n]))l++;

double w= perpendicularProjection(p[r%n], p[i], p[(i+1)%n])-perpendicularProjection(p[l%n], p[i], p[(i+1)%n]);

double h= getDistanceToLine(p[u%n], p[i], p[(i+1)%n]);

ret.first= min(ret.first, w\*h);

ret.second= min(ret.second, 2.0\*(w+h));

}

return ret;

}

void rotatingCalipers(Point \*p, int n, vector<Segment>& sol) {

sol.clear();

int j = 1; p[n] = p[0];

for (int i = 0; i < n; i++) {

while (getCross(p[j+1]-p[i+1], p[i]-p[i+1]) > getCross(p[j]-p[i+1], p[i]-p[i+1]))

j = (j+1) % n;

sol.push\_back(Segment(p[i],p[j]));

sol.push\_back(Segment(p[i + 1],p[j + 1]));

}

}

void rotatingCalipersGetRectangle (Point \*p, int n, double& area, double& perimeter) {

p[n] = p[0];

int l = 1, r = 1, j = 1;

area = perimeter = 1e20;

for (int i = 0; i < n; i++) {

Vector v = (p[i+1]-p[i]) / getLength(p[i+1]-p[i]);

while (dcmp(getDot(v, p[r%n]-p[i]) - getDot(v, p[(r+1)%n]-p[i])) < 0) r++;

while (j < r || dcmp(getCross(v, p[j%n]-p[i]) - getCross(v,p[(j+1)%n]-p[i])) < 0) j++;

while (l < j || dcmp(getDot(v, p[l%n]-p[i]) - getDot(v, p[(l+1)%n]-p[i])) > 0) l++;

double w = getDot(v, p[r%n]-p[i])-getDot(v, p[l%n]-p[i]);

double h = getDistanceToLine (p[j%n], p[i], p[i+1]);

area = min(area, w \* h);

perimeter = min(perimeter, 2 \* w + 2 \* h);

}

}

Polygon cutPolygon (Polygon u, Point a, Point b) {

Polygon ret;

int n = u.size();

for (int i = 0; i < n; i++) {

Point c = u[i], d = u[(i+1)%n];

if (dcmp((b-a)\*(c-a)) >= 0) ret.push\_back(c);

if (dcmp((b-a)\*(d-c)) != 0) {

Point t;

getIntersection(a, b-a, c, d-c, t);

if (onSegment(t, c, d))

ret.push\_back(t);

}

}

return ret;

}

int halfPlaneIntersection(DirLine\* li, int n, Point\* poly) {

sort(li, li + n);

int first, last;

Point\* p = new Point[n];

DirLine\* q = new DirLine[n];

q[first=last=0] = li[0];

for (int i = 1; i < n; i++) {

while (first < last && !onLeft(li[i], p[last-1])) last--;

while (first < last && !onLeft(li[i], p[first])) first++;

q[++last] = li[i];

if (dcmp(q[last].v \* q[last-1].v) == 0) {

last--;

if (onLeft(q[last], li[i].p)) q[last] = li[i];

}

if (first < last)

getIntersection(q[last-1].p, q[last-1].v, q[last].p, q[last].v, p[last-1]);

}

while (first < last && !onLeft(q[first], p[last-1])) last--;

if (last - first <= 1) { delete [] p; delete [] q; return 0; }

getIntersection(q[last].p, q[last].v, q[first].p, q[first].v, p[last]);

int m = 0;

for (int i = first; i <= last; i++) poly[m++] = p[i];

delete [] p; delete [] q;

return m;

}

Polygon simplify (const Polygon& poly) {

Polygon ret;

int n = poly.size();

for (int i = 0; i < n; i++) {

Point a = poly[i];

Point b = poly[(i+1)%n];

Point c = poly[(i+2)%n];

if (dcmp((b-a)\*(c-b)) != 0 && (ret.size() == 0 || b != ret[ret.size()-1]))

ret.push\_back(b);

}

return ret;

}

Point ComputeCentroid( Point\* p,int n){

Point c(0,0);

double scale = 6.0 \* getSignedArea(p,n);

for (int i = 0; i < n; i++){

int j = (i+1) % n;

c = c + (p[i]+p[j])\*(p[i].x\*p[j].y - p[j].x\*p[i].y);

}

return c / scale;

}

/// pt must be in ccw order with no three collinear points

/// returns inside = 1, on = 0, outside = -1

int pointInConvexPolygon(Point\* pt, int n, Point p){

assert(n >= 3);

int lo = 1 , hi = n - 1 ;

while(hi - lo > 1){

int mid = (lo + hi) / 2;

if(getCross(pt[mid] - pt[0], p - pt[0]) > 0) lo = mid;

else hi = mid;

}

bool in = pointInTriangle(pt[0], pt[lo], pt[hi], p);

if(!in) return -1;

if(getCross(pt[lo] - pt[lo-1], p - pt[lo-1]) == 0) return 0;

if(getCross(pt[hi] - pt[lo], p - pt[lo]) == 0) return 0;

if(getCross(pt[hi] - pt[(hi+1)%n], p - pt[(hi+1)%n]) == 0) return 0;

return 1;

}

// Calculate [ACW, CW] tangent pair from an external point

#define CW -1

#define ACW 1

int direction(Point st, Point ed, Point q) {return dcmp(getCross(ed - st, q - ed));}

bool isGood(Point u, Point v, Point Q, int dir) {return direction(Q, u, v) != -dir;}

Point better(Point u, Point v, Point Q, int dir) {return direction(Q, u, v) == dir ? u : v;}

Point tangents(Point\* pt, Point Q, int dir, int lo, int hi){

while(hi - lo > 1){

int mid = (lo + hi)/2;

bool pvs = isGood(pt[mid], pt[mid - 1], Q, dir);

bool nxt = isGood(pt[mid], pt[mid + 1], Q, dir);

if(pvs && nxt) return pt[mid];

if(!(pvs || nxt)){

Point p1 = tangents(pt, Q, dir, mid+1, hi);

Point p2 = tangents(pt, Q, dir, lo, mid - 1);

return better(p1, p2, Q, dir);

}

if(!pvs){

if(direction(Q, pt[mid], pt[lo]) == dir) hi = mid - 1;

else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid - 1;

else lo = mid + 1;

}

if(!nxt){

if(direction(Q, pt[mid], pt[lo]) == dir) lo = mid + 1;

else if(better(pt[lo], pt[hi], Q, dir) == pt[lo]) hi = mid - 1;

else lo = mid + 1;

}

}

Point ret = pt[lo];

for(int i = lo + 1; i <= hi; i++) ret = better(ret, pt[i], Q, dir);

return ret;

}

// [ACW, CW] Tangent

pair<Point, Point> get\_tangents(Point\* pt, int n, Point Q){

Point acw\_tan = tangents(pt, Q, ACW, 0, n - 1);

Point cw\_tan = tangents(pt, Q, CW, 0, n - 1);

return make\_pair(acw\_tan, cw\_tan);

}

};

struct Circle {

Point o;

double r;

Circle () {}

Circle (Point o, double r = 0): o(o), r(r) {}

void read () { o.read(), scanf("%lf", &r); }

Point point(double rad) { return Point(o.x + cos(rad)\*r, o.y + sin(rad)\*r); }

double getArea (double rad) { return rad \* r \* r / 2; }

//area of the circular sector cut by a chord with central angle alpha

double sector(double alpha) {return r \* r \* 0.5 \* (alpha - sin(alpha));}

};

namespace Circular {

using namespace Linear;

using namespace Vectorial;

using namespace Triangular;

int getLineCircleIntersection(Point p, Point q, Circle O, double& t1, double& t2, vector<Point>& sol) {

Vector v = q-p;

//sol.clear();

double a = v.x, b = p.x - O.o.x, c = v.y, d = p.y - O.o.y;

double e = a\*a+c\*c, f = 2\*(a\*b+c\*d), g = b\*b+d\*d-O.r\*O.r;

double delta = f\*f - 4\*e\*g;

if (dcmp(delta) < 0) return 0;

if (dcmp(delta) == 0) {

t1 = t2 = -f / (2 \* e);

sol.push\_back(p + v \* t1);

return 1;

}

t1 = (-f - sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t1);

t2 = (-f + sqrt(delta)) / (2 \* e); sol.push\_back(p + v \* t2);

return 2;

}

/// signed area of intersection of circle(c.o, c.r) and

/// triangle(c.o, s.a, s.b) [cross(a-o, b-o)/2]

double areaCircleTriIntersection(Circle c, Segment s){

using namespace Linear;

double OA = getLength(c.o - s.a);

double OB = getLength(c.o - s.b);

// sector

if (dcmp(getDistanceToSegment(c.o, s.a, s.b) - c.r) >= 0)

return fix\_acute(getSignedAngle(s.a - c.o, s.b - c.o)) \* (c.r\*c.r) / 2.0;

// triangle

if (dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)

return getCross(c.o-s.b,s.a-s.b) / 2.0;

// three part: (A, a) (a, b) (b, B)

vector<Point>Sect; double t1,t2;

getLineCircleIntersection(s.a, s.b, c, t1, t2, Sect);

return areaCircleTriIntersection(c, Segment(s.a, Sect[0]))

+ areaCircleTriIntersection(c, Segment(Sect[0], Sect[1]))

+ areaCircleTriIntersection(c, Segment(Sect[1], s.b));

}

// area of intersection of circle(c.o, c.r) and simple polygon(p[])

double areaCirclePolygon(Circle c, Polygon p){

double res = 0.0;

int n = p.size();

for(int i=0; i<n; ++i)

res += areaCircleTriIntersection(c, Segment(p[i], p[(i+1)%n]));

return fabs(res);

}

// interior (d < R - r) ----> -2

// interior tangents (d = R - r) ----> -1

// concentric (d = 0)

// secants (R - r < d < R + r) ----> 0

// exterior tangents (d = R + r) ----> 1

// exterior (d > R + r) ----> 2

int getPos(Circle o1, Circle o2) {

using namespace Vectorial;

double d = getLength(o1.o - o2.o);

int in = dcmp(d - fabs(o1.r - o2.r)), ex = dcmp(d - (o1.r + o2.r));

return in<0 ? -2 : in==0? -1 : ex==0 ? 1 : ex>0? 2 : 0;

}

int getCircleCircleIntersection (Circle o1, Circle o2, vector<Point>& sol) {

double d = getLength(o1.o - o2.o);

if (dcmp(d) == 0) {

if (dcmp(o1.r - o2.r) == 0) return -1;

return 0;

}

if (dcmp(o1.r + o2.r - d) < 0) return 0;

if (dcmp(fabs(o1.r-o2.r) - d) > 0) return 0;

Vector v = o2.o - o1.o;

double co = (o1.r\*o1.r + getPLength(v) - o2.r\*o2.r) / (2 \* o1.r \* getLength(v));

double si = sqrt(fabs(1.0 - co\*co));

Point p1 = scale(cw(v,co, si), o1.r) + o1.o;

Point p2 = scale(ccw(v,co, si), o1.r) + o1.o;

sol.push\_back(p1);

if (p1 == p2) return 1;

sol.push\_back(p2);

return 2;

}

double areaCircleCircle(Circle o1, Circle o2){

Vector AB = o2.o - o1.o;

double d = getLength(AB);

if(d >= o1.r + o2.r) return 0;

if(d + o1.r <= o2.r) return pi \* o1.r \* o1.r;

if(d + o2.r <= o1.r) return pi \* o2.r \* o2.r;

double alpha1 = acos((o1.r \* o1.r + d \* d - o2.r \* o2.r) / (2.0 \* o1.r \* d));

double alpha2 = acos((o2.r \* o2.r + d \* d - o1.r \* o1.r) / (2.0 \* o2.r \* d));

return o1.sector(2\*alpha1) + o2.sector(2\*alpha2);

}

int getTangents (Point p, Circle o, Vector\* v) {

Vector u = o.o - p;

double d = getLength(u);

if (d < o.r) return 0;

else if (dcmp(d - o.r) == 0) {

v[0] = rotate(u, pi / 2);

return 1;

} else {

double ang = asin(o.r / d);

v[0] = rotate(u, -ang);

v[1] = rotate(u, ang);

return 2;

}

}

int getTangentPoints (Point p, Circle o, vector<Point>& v) {

Vector u = p - o.o ;

double d = getLength(u);

if (d < o.r) return 0;

else if (dcmp(d - o.r) == 0) {

v.push\_back(o.o+u);

return 1;

} else {

double ang = acos(o.r / d);

u = u / getLength(u) \* o.r;

v.push\_back(o.o+rotate(u, -ang));

v.push\_back(o.o+rotate(u, ang));

return 2;

}

}

int getTangents (Circle o1, Circle o2, Point\* a, Point\* b) {

int cnt = 0;

if (dcmp(o1.r-o2.r) < 0) { swap(o1, o2); swap(a, b); }

double d2 = getPLength(o1.o - o2.o);

double rdif = o1.r - o2.r, rsum = o1.r + o2.r;

if (dcmp(d2 - rdif \* rdif) < 0) return 0;

if (dcmp(d2) == 0 && dcmp(o1.r - o2.r) == 0) return -1;

double base = getAngle(o2.o - o1.o);

if (dcmp(d2 - rdif \* rdif) == 0) {

a[cnt] = o1.point(base); b[cnt] = o2.point(base); cnt++;

return cnt;

}

double ang = acos( (o1.r - o2.r) / sqrt(d2) );

a[cnt] = o1.point(base+ang); b[cnt] = o2.point(base+ang); cnt++;

a[cnt] = o1.point(base-ang); b[cnt] = o2.point(base-ang); cnt++;

if (dcmp(d2 - rsum \* rsum) == 0) {

a[cnt] = o1.point(base); b[cnt] = o2.point(pi+base); cnt++;

}

else if (dcmp(d2 - rsum \* rsum) > 0) {

double ang = acos( (o1.r + o2.r) / sqrt(d2) );

a[cnt] = o1.point(base+ang); b[cnt] = o2.point(pi+base+ang); cnt++;

a[cnt] = o1.point(base-ang); b[cnt] = o2.point(pi+base-ang); cnt++;

}

return cnt;

}

Circle CircumscribedCircle(Point p1, Point p2, Point p3) {

double Bx = p2.x - p1.x, By = p2.y - p1.y;

double Cx = p3.x - p1.x, Cy = p3.y - p1.y;

double D = 2 \* (Bx \* Cy - By \* Cx);

double cx = (Cy \* (Bx \* Bx + By \* By) - By \* (Cx \* Cx + Cy \* Cy)) / D + p1.x;

double cy = (Bx \* (Cx \* Cx + Cy \* Cy) - Cx \* (Bx \* Bx + By \* By)) / D + p1.y;

Point p = Point(cx, cy);

return Circle(p, getLength(p1 - p));

}

Circle InscribedCircle(Point p1, Point p2, Point p3) {

double a = getLength(p2 - p3);

double b = getLength(p3 - p1);

double c = getLength(p1 - p2);

Point p = (p1 \* a + p2 \* b + p3 \* c) / (a + b + c);

return Circle(p, getDistanceToLine(p, p1, p2));

}

/// distance From P : distance from Q = rp : rq

Circle getApolloniusCircle(const Point& P,const Point& Q, double rp, double rq ){

rq \*= rq ;

rp \*= rp ;

double a = rq - rp ;

assert(dcmp(a));

double g = rq \* P.x - rp \* Q.x ; g /= a ;

double h = rq \* P.y - rp \* Q.y ; h /= a ;

double c = rq\*P.x\*P.x-rp\*Q.x\*Q.x+rq\*P.y\*P.y-rp\*Q.y\*Q.y ;

c /= a ;

Point o(g,h);

double R = g\*g +h\*h - c ;

R = sqrt(R);

return Circle(o,R);

}

};

struct Star{

int n; /// number of side of the star

double r; /// radius of the circum-circle

Star(int n,double r) {this->n=n; this->r=r;}

double getArea(){

double theta=pi/n;

double s=2\*r\*sin(theta);

double R=0.5\*s/tan(theta);

double a=0.5\*n\*s\*R;

double a2=0.25\*s\*s/tan(1.5\*theta);

return a-n\*a2;

}

};

● **PBDS:**

#include <ext/pb\_ds/tree\_policy.hpp>

#include <ext/pb\_ds/assoc\_container.hpp>

using namespace \_\_gnu\_pbds;

template<typename temp>using ordered\_set = tree<temp, null\_type, less<temp>, rb\_tree\_tag,tree\_order\_statistics\_node\_update>;

● **Unordered Map:**

struct custom\_hash {

static uint64\_t splitmix64(uint64\_t x) {

x += 0x9e3779b97f4a7c15; x = (x ^ (x >> 30)) \* 0xbf58476d1ce4e5b9;

x = (x ^ (x >> 27)) \* 0x94d049bb133111eb;

return x ^ (x >> 31);

}

size\_t operator()(uint64\_t x) const {

static const uint64\_t FIXED\_RANDOM = chrono::steady\_clock::now().time\_since\_epoch().count();

return splitmix64(x + FIXED\_RANDOM);

}

};

unordered\_map<long long, int, custom\_hash> safe\_map;

**Combinatorics**: **Distribution:**

**distinct m ball in distinct n box** -> n \* ( T(m-1, n-1) + T(m-1, n))

**distinct m ball in identical n box** -> S(m-1, n-1) + n \* S(m-1, n)

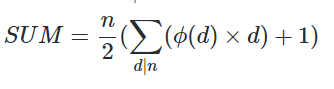
**Box in circular / Flag pole with identical pole** ->

S(m-1, n-1)+(m-1)\*S(m-1,n)

**identcal m ball in identical n box** ->. P(m-1, n-1) + n \* P(m-n, n)

**identical m ball in identical n box** -> (m-1)c(n-1)

**Sum of LCM**





**Derangement:** D(n)= n\*D(n-1)+(-1)^n

**Algebra Formula:** GCD((x^a)-1, (x^b)-1)= (x^GCD(a, b))-1

**A-dominating Sequence: n As, m Bs(n>=m)**

**Number of Dominating seq:** (n+m)Cn - (n+m)C(n+1)

**Number of spanning tree in Bipartite Graph:**

G(X, Y)= X^(Y-1) \* Y^(X-1)

**Where X is number of nodes in 1st set, Y is in 2nd set.**

**Random Function:**

Mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count())

**// for int64, use mt19937\_64**

int rand\_func(int l, int r){

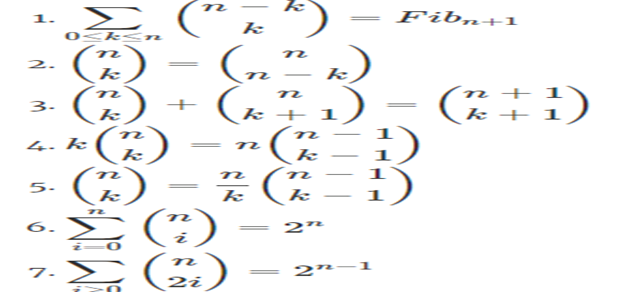
return uniform\_int\_distribution<int>(l, r) (rng);}

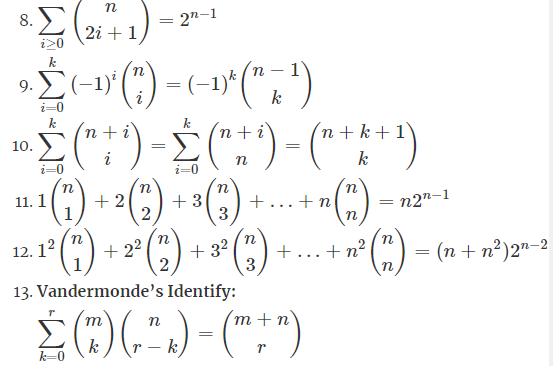
**Random Prime:**

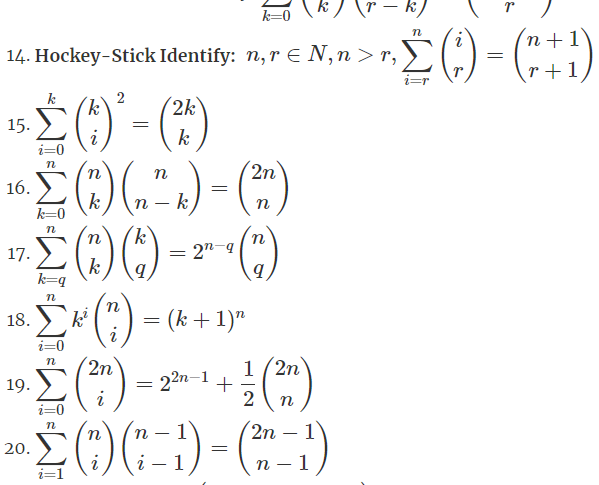
1500450271, 3267000013, 4093082899, 3628273133,

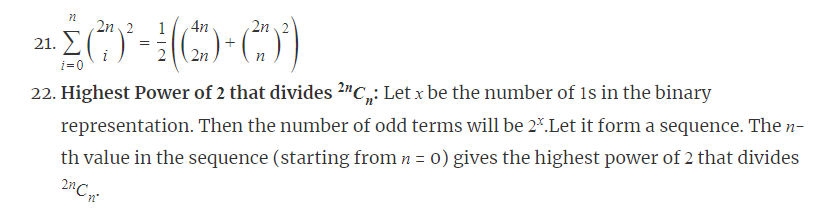
2860486313, 3367900313

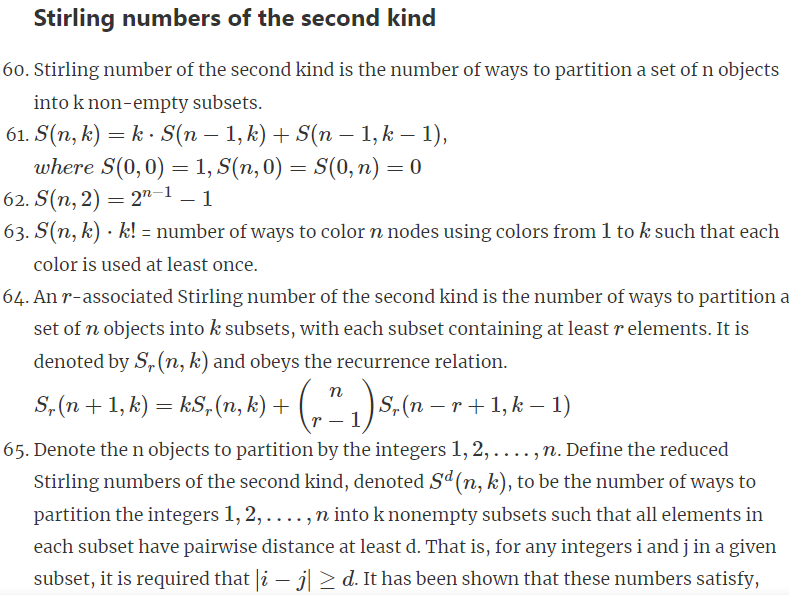
**Binomial Coefficient property:**

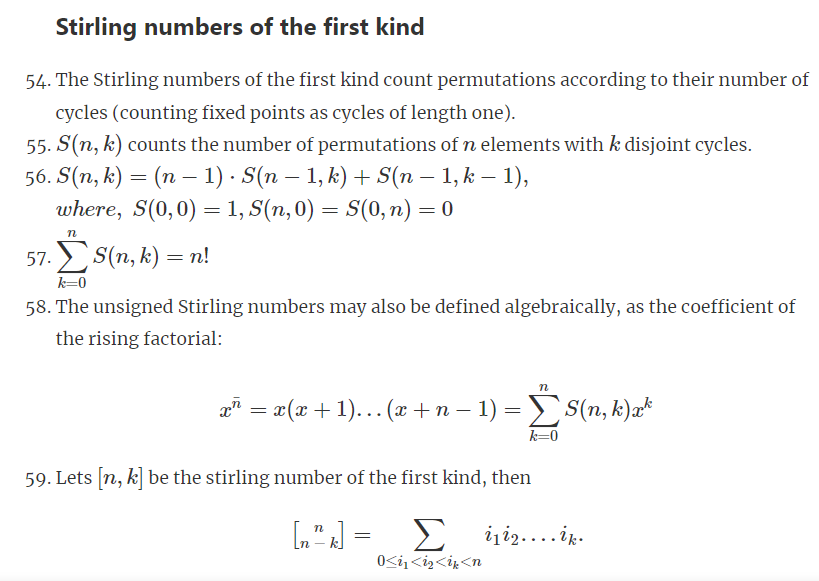


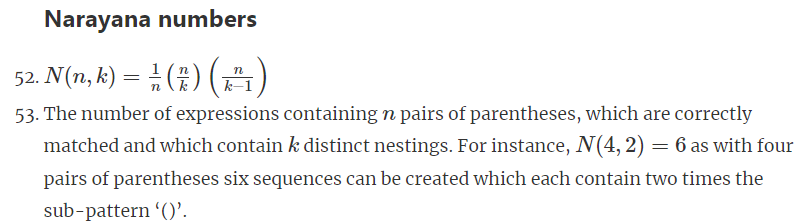


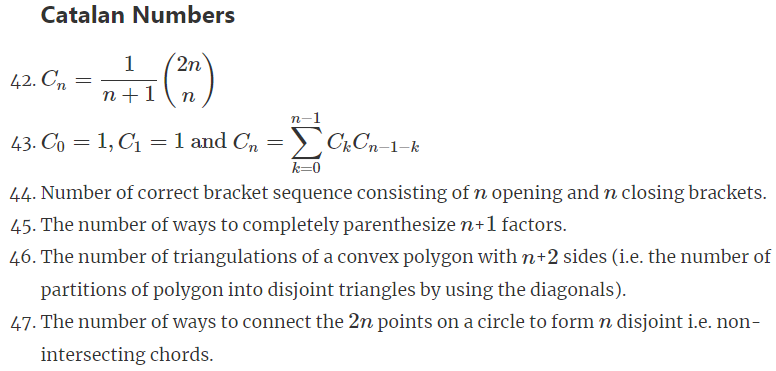


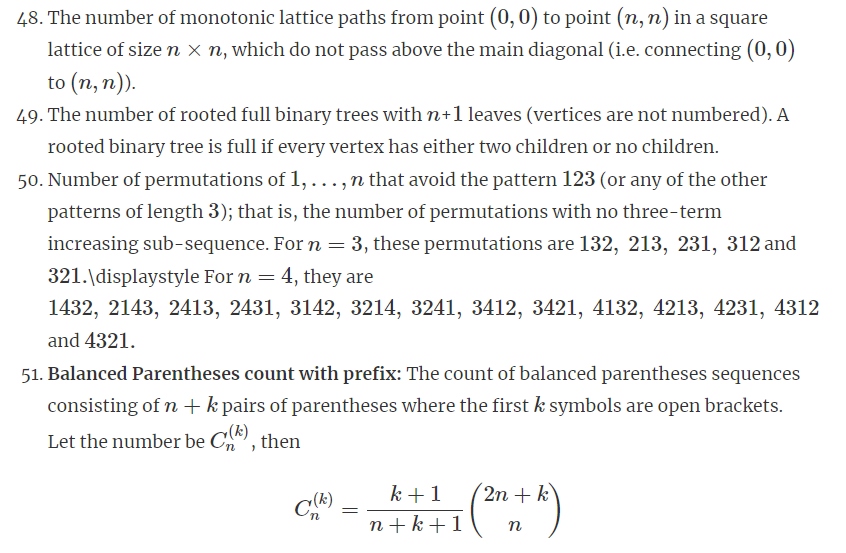


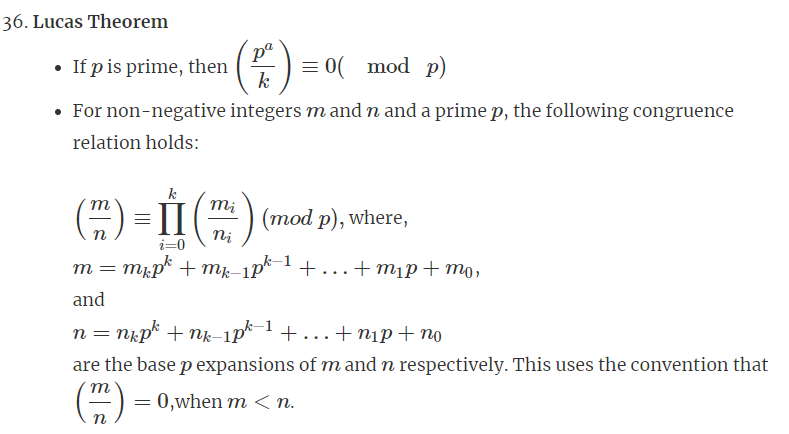
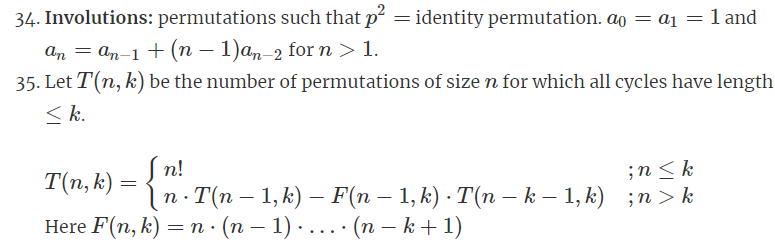
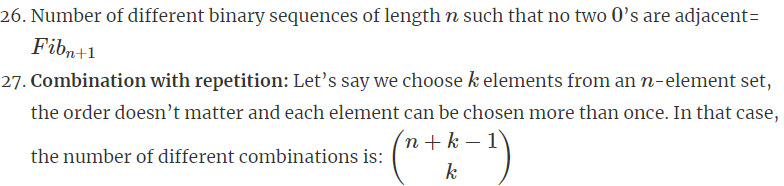


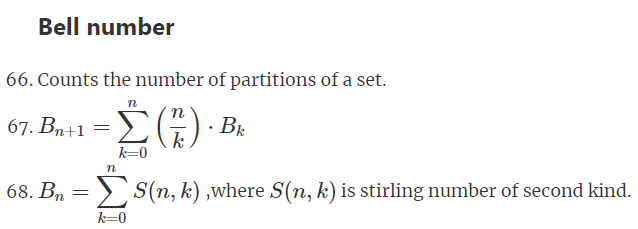










**Segment Tree Beats:**

Keep min & max do update when min==max.