CygnusAlpha ACM-ICPC Notebook 2018 (C++)

Contents

1 Geometry 1
1.1 Convex Hull 1
1.2 Point inside polygon 1
1.3 Welzian algo 2

2 Graphs 2
2.1 Articulation and Bridge points 2
2.2 Dijkstra 3
2.3 LCA 3

3 Flows 3
3.1 Bipartite 3
3.2 Max-Flow 4
3.3 MCMF 6

4 Tree 7
4.1 BIT 7
4.2 Segment Tree 8
4.3 Lazy-Segment Tree 8
4.4 Policy Tree 10
4.5 Trie 10

5 Math 11
5.1 CRT 11
5.2 DigitDP 11
5.3 DP DNC 11
5.4 Euclidean 12
5.5 Factors in n-1-3 12
5.6 Fibo logn 13
5.7 EGaussian Algorithm 13
5.8 Lucas theorem 13
5.9 Matrix expo 14
5.10 Miller-Rabin 14
5.11 Mobius 14
5.12 SQRT CBRT tourist 15
5.13 Euler totient 15
5.14 FFT 15
5.15 FFT-DNC 16

6 Strings 17
6.1 Knuth-Morris-Pratt Algorithm 17
6.2 Suffix array 18
6.3 z-function 18

7 EZPZ 18
7.1 Template 18
7.2 fast io 19
7.3 LIS nlogn 19
7.4 MOs 19

1 Geometry

1.1 Convex Hull

```
//convex hull
typedef pair<ll,ll> point;
ll cross (point a, point b, point c) { return (b.x - a.x
    ) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);}
vector<point> ConvexHull(vector<point>&p, ll n) {
    ll sz = 0;
    vector<point> hull(n + n);
    sort(p.begin(), p.end());
    for(ll i = 0; i < n; ++i) {
        while (sz > 1 and cross(hull[sz - 2], hull[sz -
            1], p[i]) <= 0) --sz;
        hull[sz++] = p[i];
    }
    for(ll i = n - 2, j = sz + 1; i >= 0; --i) {
        while (sz >= j and cross(hull[sz - 2], hull[sz -
             1], p[i]) <= 0) --sz;
        hull[sz++] = p[i];
    } hull.resize(sz - 1);
    return hull;
}
```

1.2 Point inside polygon

```
//points inside convex polygon O(logn)
const ll N = 100009;
struct point {
    ll x,y;
}a[N];
ll n;
double cross(const point& p1,const point& p2,const point
    & org) {
    return ((p1.x-org.x)*1.0)*(p2.y-org.y)-((p2.x-org.x)
        *1.0)*(p1.y-org.y);
}
inline bool comp(const point& x,const point& y) {
    return cross(x,y,a[0]) >= 0;
}
bool inside(point& p) {
    if (cross(a[0], a[n-1], p)>=0) return false;
    if (cross(a[0], a[1], p) <=0) return false;
    ll l =1;
    ll r =n-1;
    while(l<r) {
        ll m = l + (r-l)/2;
        if(cross(a[m],p,a[0]) >=0)
            l=m+1;
        else
            r=m;
    }
    if(l == 0)
        return false;
    return cross(a[l-1],a[l],p) >0;
}
sort(a+1,a+n,comp);
```

1.3 Welzian algo

```
//welzian algo
struct point {
    long double x;
    long double y;
};
struct circle {
    long double x;
    long double y;
    long double r;
    circle() {}
    circle(long double x, long double y, long double r):
        x(x), y(y), r(r) {}
};
circle b_md(vector<point> R) {
    if (R.size() == 0) {
        return circle(0, 0, -1);
    } else if (R.size() == 1) {
        return circle(R[0].x, R[0].y, 0);
    } else if (R.size() == 2) {
        return circle((R[0].x+R[1].x)/2.0, (R[0].y+R[1].
            y)/2.0,hypot(R[0].x-R[1].x, R[0].y-R[1].y)
            /2.0);
    } else {
        long double D = (R[0].x - R[2].x)*(R[1].y - R
            [2].y) - (R[1].x - R[2].x)*(R[0].y - R[2].y)
            ;
        long double p0 = (((R[0].x - R[2].x)*(R[0].x + R
            [2].x) + (R[0].y - R[2].y)*(R[0].y + R[2].y)
            ) / 2 * (R[1].y - R[2].y) - ((R[1].x - R[2].
            x)*(R[1].x + R[2].x) + (R[1].y - R[2].y)*(R
            [1].y + R[2].y)) / 2 * (R[0].y - R[2].y))/D;
        long double p1 = (((R[1].x - R[2].x)*(R[1].x + R
            [2].x) + (R[1].y - R[2].y)*(R[1].y + R[2].y)
            ) / 2 * (R[0].x - R[2].x) - ((R[0].x - R[2].
            x)*(R[0].x + R[2].x) + (R[0].y - R[2].y)*(R
            [0].y + R[2].y)) / 2 * (R[1].x - R[2].x))/D;
        return circle(p0, p1, hypot(R[0].x - p0, R[0].y
            - p1));
    }
}
circle b_minidisk(vector<point>& P, int i, vector<point>
    R) {
    if (i == P.size() || R.size() == 3) {
        return b_md(R);
    } else {
        circle D = b_minidisk(P, i+1, R);
        if (hypot(P[i].x-D.x, P[i].y-D.y) > D.r) {
            R.push_back(P[i]);
            D = b_minidisk(P, i+1, R);
        }
        return D;
    }
}
// Call this function.
circle minidisk(vector<point> P) {
    random_shuffle(P.begin(), P.end());
    return b_minidisk(P, 0, vector<point>());
}
```

2 Graphs

2.1 Articulation and Bridge points

```
vector<ll>v[100003];
ll disc[100003];
ll low[100003];
ll vis[100003];
set<ll>ap;
set<pair<ll,ll>>br;
ll par[100003];

void dfs(ll p, ll t){
    vis[p]=1;
    disc[p]=t+1;
    low[p]=t+1;
    ll ch=0;
    for(auto e:v[p]){
        if(vis[e]==0){
            ch++;
            par[e]=p;
            dfs(e,t+1);
            low[p]=min(low[p],low[e]);
            if(low[e]>disc[p]){
                br.insert(mp(min(e,p),max(e,p)));
            }
            if(par[p]==0 && ch>1){
                ap.insert(p);
            }else if(par[p]!=0){
                if(low[e]>=disc[p]){
                    ap.insert(p);
                }
            }
        }else if(e!=par[p]){
            low[p]=min(low[p],disc[e]);
        }
    }
```

```
}

int main()
{
        par[1]=0;
        dfs(1,0);
}
```

2.2 Dijkstra

```
vector<pair<ll,ll> >v[100003];
ll dist[100003];
int main(){
    ios::sync_with_stdio(0);
    ll n,e,a[1000003];
    cin>>n>>e;
    for(ll i=0;i<e;i++){
        ll p,q,w;
        cin>>p>>q>>w;
        v[p].pb(mp(w,q));
        v[q].pb(mp(w,p));
    }
    for(ll i=0;i<=n;i++){
        dist[i]=100000;
    }
    ll so;
    cin>>so;
    set<pair<ll,ll> >s;
    dist[so]=0;
    s.insert(mp(0,so));
    while(!s.empty()){
        pair<ll,ll>p=*(s.begin());
        s.erase(p);
        for(ll i=0;i<v[p.se].size();i++){
            if(dist[v[p.se][i].se]>dist[p.se]+v[p.se][i
                ].fi){
                if(dist[v[p.se][i].se]!=100000){
                    s.erase(v[p.se][i]);
                }
                dist[v[p.se][i].se]=dist[p.se]+v[p.se][i
                    ].fi;
                s.insert(mp(dist[v[p.se][i].se],v[p.se][
                    i].se));
            }
        }
    }
}
```

2.3 LCA

```
int parent[MAXN], depth[MAXN], f[MAXN][LOGN + 1];
vector <int> adj[MAXN];
void dfs(int u) {
    if (u != 1) {
        f[u][0] = parent[u];
        for (int i = 1; i <= LOGN; i++)
            f[u][i] = f[f[u][i - 1]][i - 1];
    }
    for (int i = 0; i < (int) adj[u].size(); i++) {
        int v = adj[u][i];
        if (parent[v] == 0) {
            parent[v] = u;
            depth[v] = depth[u] + 1;dfs(v);
        }
    }
}
int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    for (int i = LOGN; i >= 0; i--)
        if (depth[f[u][i]] >= depth[v]) u = f[u][i];
    if (u == v) return v;
    for (int i = LOGN; i >= 0; i--)
        if (f[u][i] != f[v][i])
    u = f[u][i],v = f[v][i];
    return f[u][0];
}
```

3 Flows

3.1 Bipartite

```
//Hopcroft_Karp O(EsqrtV)
struct edge
{
        int from, to, cap, flow, index;
        edge(int from, int to, int cap, int flow, int
            index):
                from(from), to(to), cap(cap), flow(flow)
                    , index(index) {}
};

struct Hopcroft_Karp
{
static const int inf = 1e9;

int n;
```

```
vector<int> matchL, matchR, dist;
vector<vector<int> > g;

Hopcroft_Karp(int n) :
        n(n), matchL(n+1), matchR(n+1), dist(n+1), g(n
            +1) {}

void addEdge(int u, int v)
{
        g[u].push_back(v);
}

bool bfs()
{
        queue<int> q;
        for(int u=1;u<=n;u++)
        {
                if(!matchL[u])
                {
                        dist[u]=0;
                        q.push(u);
                }
                else
                        dist[u]=inf;
        }
        dist[0]=inf;

        while(!q.empty())
        {
        int u=q.front();
        q.pop();
        for(auto v:g[u])
        {
                if(dist[matchR[v]] == inf)
                {
                        dist[matchR[v]] = dist[u] + 1;
                        q.push(matchR[v]);
                }
        }
        }

        return (dist[0]!=inf);
}

bool dfs(int u)
{
        if(!u)
                return true;
        for(auto v:g[u])
        {
        if(dist[matchR[v]] == dist[u]+1 &&dfs(matchR[v])
            )
        {
                matchL[u]=v;
                matchR[v]=u;
                return true;
        }
        }
        dist[u]=inf;
        return false;
}

int max_matching()
{
        int matching=0;
        while(bfs())
        {
        for(int u=1;u<=n;u++)
        {
                if(!matchL[u])
                        if(dfs(u))
                                matching++;
        }
        }
        return matching;
}
};
int main(){
    Hopcroft_Karp mx(n+m+3);
        mx.addEdge(q,r);
        cout<<mx.max_matching()<<"\n";
}
```

3.2 Max-Flow

```
//Push relabel V^2sqrtE
struct edge
{
        int from, to, cap, flow, index;
        edge(int from, int to, int cap, int flow, int
            index):
                from(from), to(to), cap(cap), flow(flow)
                    , index(index) {}
};

struct PushRelabel
{
static const long long INF=1e18;

int n;
```

```cpp
vector<vector<edge> > g;
vector<long long> excess;
vector<int> height;

PushRelabel(int n):
        n(n), g(n), excess(n), height(n) {}

void addEdge(int from, int to, int cap)
{
        g[from].push_back(edge(from, to, cap, 0, g[to].
            size()));
        if(from==to)
                g[from].back().index++;
        g[to].push_back(edge(to, from, 0, 0, g[from].
            size()-1));
}

void push(edge &e)
{
    int amt=(int)min(excess[e.from], (long long)e.cap -
        e.flow);
        if(height[e.from]<=height[e.to] || amt==0)
                return;
        e.flow += amt;
        g[e.to][e.index].flow -= amt;
        excess[e.to] += amt;
        excess[e.from] -= amt;
}

void relabel(int u)
{
        int d=2e5;
        for(auto &it:g[u])
        {
                if(it.cap-it.flow>0)
                        d=min(d, height[it.to]);
        }
        if(d<INF)
                height[u]=d+1;
}

vector<int> find_max_height_vertices(int source, int
    dest)
{
        vector<int> max_height;
        for(int i=0;i<n;i++)
        {
        if(i!=source && i!=dest && excess[i]>0)
        {
        if(!max_height.empty() && height[i] > height[
            max_height[0]])
                max_height.clear();
        if(max_height.empty() || height[i] == height[
            max_height[0]])
                max_height.push_back(i);
        }
        }
        return max_height;
}

long long max_flow(int source, int dest)
{
        excess.assign(n, 0);
        height.assign(n, 0);
        height[source]=n;
        excess[source]=INF;
        for(auto &it:g[source])
                push(it);

        vector<int> current;
        while(!(current = find_max_height_vertices(
            source, dest)).empty())
        {
        for(auto i:current)
        {
                bool pushed=false;
                for(auto &e:g[i])
                {
                if(excess[i]==0)
                        break;
                if(e.cap - e.flow>0 && height[e.from] ==
                    height[e.to] + 1)
                {
                        push(e);
                        pushed=true;
                }
                }
                if(!pushed)
                {
                        relabel(i);
                        break;
                }
        }
        }

        long long max_flow=0;
        for(auto &e:g[source])
                max_flow+=e.flow;

        return max_flow;
```

```
    }
};

// vector<ll>v[100003];

int main(){
    bolt;
    ll n;
    cin>>n;
    map<char,ll>m;
    m['A']=0;
    m['Z']=1;
    ll ind=2;
    // ll gr[100][100]={0};
        PushRelabel mx(1000);
    forr(i,0,n){
        char a,b;
        ll len;
        cin>>a>>b>>len;
        if(m.count(a)==0){
            m[a]=ind++;
        }
        if(m.count(b)==0){
            m[b]=ind++;
        }
                // gr[m[a]][m[b]]=len;
        // gr[m[b]][m[a]]=len;
                mx.addEdge(m[a],m[b],len);
    }
        cout<<mx.max_flow(0,1)<<"\n";
}
```

3.3 MCMF

```
//Works for negative costs, but does not work for
   negative cycles
//Complexity: O(min(E^2 *V log V, E logV * flow))
struct edge
{
        int to, flow, cap, cost, rev;
};

struct MinCostMaxFlow
{
int nodes;
vector<int> prio, curflow, prevedge, prevnode, q, pot;
vector<bool> inqueue;
vector<vector<edge> > graph;
MinCostMaxFlow() {}

MinCostMaxFlow(int n): nodes(n), prio(n, 0), curflow(n,
   0),
prevedge(n, 0), prevnode(n, 0), q(n, 0), pot(n, 0),
   inqueue(n, 0), graph(n) {}

void addEdge(int source, int to, int capacity, int cost)
{
        edge a = {to, 0, capacity, cost, (int)graph[to].
           size()};
        edge b = {source, 0, 0, -cost, (int)graph[source
           ].size()};
        graph[source].push_back(a);
        graph[to].push_back(b);
}

void bellman_ford(int source, vector<int> &dist)
{
        fill(dist.begin(), dist.end(), INT_MAX);
        dist[source] = 0;
        int qt=0;
        q[qt++] = source;
        for(int qh=0;(qh-qt)%nodes!=0;qh++)
        {
        int u = q[qh%nodes];
        inqueue[u] = false;
        for(auto &e : graph[u])
        {
                if(e.flow >= e.cap)
                        continue;
                int v = e.to;
                int newDist = dist[u] + e.cost;
                if(dist[v] > newDist)
                {
                        dist[v] = newDist;
                        if(!inqueue[v])
                        {
                                inqueue[v] = true;
                                q[qt++ % nodes] = v;
                        }
                }
        }
        }
}

pair<int, int> minCostFlow(int source, int dest, int
   maxflow)
{
bellman_ford(source, pot);
int flow = 0;
int flow_cost = 0;
```

```
while(flow < maxflow)
{
        priority_queue<pair<int, int>, vector<pair<int,
            int> >, greater<pair<int, int> > > q;
        q.push({0, source});
        fill(prio.begin(), prio.end(), INT_MAX);
        prio[source] = 0;
        curflow[source] = INT_MAX;
        while(!q.empty())
        {
                int d = q.top().first;
                int u = q.top().second;
                q.pop();
                if(d != prio[u])
                        continue;
                for(int i=0;i<graph[u].size();i++)
                {
                edge &e=graph[u][i];
                int v = e.to;
                if(e.flow >= e.cap)
                        continue;
                int newPrio = prio[u] + e.cost + pot[u]
                    - pot[v];
                if(prio[v] > newPrio)
                {
                        prio[v] = newPrio;
                        q.push({newPrio, v});
                        prevnode[v] = u;
                        prevedge[v] = i;
                        curflow[v] = min(curflow[u], e.
                            cap - e.flow);
                }
                }
        }
        if(prio[dest] == INT_MAX)
                break;
        for(int i=0;i<nodes;i++)
                pot[i]+=prio[i];
        int df = min(curflow[dest], maxflow - flow);
        flow += df;
        for(int v=dest;v!=source;v=prevnode[v])
        {
                edge &e = graph[prevnode[v]][prevedge[v
                    ]];
                e.flow += df;
                graph[v][e.rev].flow -= df;
                flow_cost += df * e.cost;
        }
}
return {flow, flow_cost};
}
};
```

4 Tree

4.1 BIT

```
1D BIT:

int bit[N];

void update(int idx, int val)
{
        while(idx<=n)
        {
                bit[idx]+=val;
                idx+=idx&-idx;
        }
}

int pref(int idx)
{
        int ans=0;
        while(idx>0)
        {
                ans+=bit[idx];
                idx-=idx&-idx;
        }
        return ans;
}

int rsum(int l, int r)
{
        return pref(r) - pref(l-1);
}

Multiple BIT:

int bit[2][N];

void update(int i, int idx, int k)
{
        while(idx<=n)
        {
                bit[i][idx]+=k;
                idx+=idx&-idx;
        }
}
```

```
int pref(int i, int idx)
{
        int ans=0;
        while(idx>0)
        {
                ans+=bit[i][idx];
                idx-=idx&-idx;
        }
        return ans;
}

int rsum(int i, int l, int r)
{
        return pref(i, r) - pref(i, l-1);
}
```

4.2 Segment Tree

```
void build(ll node,ll a, ll b){//1,0,n-1
    if(a>b)
        return;
    if(a==b){
        tree[node]=arr[a];//something
        return;
    }
    build(node*2, a, (a+b)/2);
    build(node*2+1, 1+(a+b)/2, b);
    tree[node] = tree[node*2]+tree[node*2+1]//something
}

ll query(ll node, ll a, ll b, ll i, ll j){//a=0,b=n-1,i=
   l,j=r
    if(a > b || a > j || b < i)
        return 0;
    if(a >= i && b <= j){
        return 0;//something
    }
    ll q1 = query(node*2, a, (a+b)/2, i, j);
    ll q2 = query(1+node*2, 1+(a+b)/2, b, i, j);
    return 0;//something
}

ll update(ll node, ll a, ll b, ll i, ll val){
    if(a==b){
        arr[i]=val;
        tree[node];//something
    }
    else{
        ll mid=(a+b)/2;
        if(a<=i&&i<=mid){
            update(2*node,a,mid,i,val);
        }
        else{
            update(2*node+1,mid+1,b,i,val);
        }
        tree[node]=(tree[2*node]+tree[2*node+1])%mod;//
           something
    }
}
```

4.3 Lazy-Segment Tree

```
int tree[MAX] = {0};  // To store segment tree
int lazy[MAX] = {0};  // To store pending updates

/*  si -> index of current node in segment tree
    ss and se -> Starting and ending indexes of elements
        for
                which current nodes stores sum.
    us and ue -> starting and ending indexes of update
       query
    diff -> which we need to add in the range us to ue
       */
void updateRangeUtil(int si, int ss, int se, int us,
                     int ue, int diff)
{
    // If lazy value is non-zero for current node of
       segment
    // tree, then there are some pending updates. So we
       need
    // to make sure that the pending updates are done
       before
    // making new updates. Because this value may be
       used by
    // parent after recursive calls (See last line of
       this
    // function)
    if (lazy[si] != 0)
    {
        // Make pending updates using value stored in
           lazy
        // nodes
        tree[si] += (se-ss+1)*lazy[si];

        // checking if it is not leaf node because if
        // it is leaf node then we cannot go further
        if (ss != se)
        {
            // We can postpone updating children we don'
               t
```

```
            // need their new values now.
            // Since we are not yet updating children of
                 si,
            // we need to set lazy flags for the
               children
            lazy[si*2 + 1]   += lazy[si];
            lazy[si*2 + 2]   += lazy[si];
        }

        // Set the lazy value for current node as 0 as
            it
        // has been updated
        lazy[si] = 0;
    }

    // out of range
    if (ss>se || ss>ue || se<us)
        return ;

    // Current segment is fully in range
    if (ss>=us && se<=ue)
    {
        // Add the difference to current node
        tree[si] += (se-ss+1)*diff;

        // same logic for checking leaf node or not
        if (ss != se)
        {
            // This is where we store values in lazy
               nodes,
            // rather than updating the segment tree
               itelf
            // Since we don't need these updated values
               now
            // we postpone updates by storing values in
               lazy[]
            lazy[si*2 + 1]   += diff;
            lazy[si*2 + 2]   += diff;
        }
        return;
    }

    // If not completely in rang, but overlaps, recur
       for
    // children,
    int mid = (ss+se)/2;
    updateRangeUtil(si*2+1, ss, mid, us, ue, diff);
    updateRangeUtil(si*2+2, mid+1, se, us, ue, diff);

    // And use the result of children calls to update
        this
    // node
    tree[si] = tree[si*2+1] + tree[si*2+2];
}

// Function to update a range of values in segment
// tree
/*  us and eu -> starting and ending indexes of update
   query
    ue  -> ending index of update query
    diff -> which we need to add in the range us to ue
         */
void updateRange(int n, int us, int ue, int diff)
{
   updateRangeUtil(0, 0, n-1, us, ue, diff);
}


/*  A recursive function to get the sum of values in
   given
    range of the array. The following are parameters for
    this function.
    si --> Index of current node in the segment tree.
           Initially 0 is passed as root is always at'
           index 0
    ss & se  --> Starting and ending indexes of the
                 segment represented by current node,
                 i.e., tree[si]
    qs & qe  --> Starting and ending indexes of query
                 range */
int getSumUtil(int ss, int se, int qs, int qe, int si)
{
    // If lazy flag is set for current node of segment
         tree,
    // then there are some pending updates. So we need
         to
    // make sure that the pending updates are done
         before
    // processing the sub sum query
    if (lazy[si] != 0)
    {
        // Make pending updates to this node. Note that
             this
        // node represents sum of elements in arr[ss..se
             ] and
        // all these elements must be increased by lazy[
             si]
        tree[si] += (se-ss+1)*lazy[si];

        // checking if it is not leaf node because if
```

```
        // it is leaf node then we cannot go further
        if (ss != se)
        {
            // Since we are not yet updating children os
                si,
            // we need to set lazy values for the
               children
            lazy[si*2+1] += lazy[si];
            lazy[si*2+2] += lazy[si];
        }

        // unset the lazy value for current node as it
           has
        // been updated
        lazy[si] = 0;
    }

    // Out of range
    if (ss>se || ss>qe || se<qs)
        return 0;

    // At this point we are sure that pending lazy
       updates
    // are done for current node. So we can return value
    // (same as it was for query in our previous post)

    // If this segment lies in range
    if (ss>=qs && se<=qe)
        return tree[si];

    // If a part of this segment overlaps with the given
    // range
    int mid = (ss + se)/2;
    return getSumUtil(ss, mid, qs, qe, 2*si+1) +
           getSumUtil(mid+1, se, qs, qe, 2*si+2);
}
```

4.4 Policy Tree

```
// policy tree (for o(1) dist in set)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
typedef tree<int ,null_type,less<int >,rb_tree_tag,
   tree_order_statistics_node_update>ordered_set;
xx=(m[v].order_of_key(r+1))-(m[v].order_of_key(l)); //
   dist bw with l and r
```

4.5 Trie

```
struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];
    // isEndOfWord is true if the node represents
    // end of a word
    bool isEndOfWord;
};

// Returns new trie node (initialized to NULLs)
struct TrieNode *getNode(void)
{
    struct TrieNode *pNode =  new TrieNode;
    pNode->isEndOfWord = false;
    for (int i = 0; i < ALPHABET_SIZE; i++)
        pNode->children[i] = NULL;
    return pNode;
}

// If not present, inserts key into trie
// If the key is prefix of trie node, just
// marks leaf node
void insert(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;
    for (int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();
        pCrawl = pCrawl->children[index];
    }
    // mark last node as leaf
    pCrawl->isEndOfWord = true;
}

// Returns true if key presents in trie, else
// false
bool search(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;
    for (int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            return false;
        pCrawl = pCrawl->children[index];
    }
    return (pCrawl != NULL && pCrawl->isEndOfWord);
}
```

```
// Driver
int main()
{
    string keys[] = {"the", "a", "there",
                    "answer", "any", "by",
                     "bye", "their" };
    int n = sizeof(keys)/sizeof(keys[0]);
    struct TrieNode *root = getNode();
}
```

5 Math

5.1 CRT

```
//crt
ll crt(vector<ll> v, vector<ll> rem){
        int n=v.size();
        ll M=1;
        for(auto e:rem){
                M*=e;
        }
        ll ans=0;
        for(int i=0;i<n;i++){
                ans=ans+v[i]*(inv(M/rem[i],rem[i])*M/rem
                    [i])%M;
                ans=ans%M;
        }
        return ans;
}
```

5.2 DigitDP

```
vector<int> dig; // contains digits of number
ll dp[24][204][2];
ll get(int pos,int sum,int flag){ //flag checking length
     of prefix
        if(pos==dig.size()){
                if(!pr[sum]){ // end condition
                        return 1;
                }
                else return 0;
        }
        if(dp[pos][sum][flag]!=-1){
                return dp[pos][sum][flag];
        }
        int lmt;
        ll ans=0;
        if(!flag){
                lmt=dig[pos];
        }else{
                lmt=9;
        }
        for(int i=0;i<=lmt;i++){
                int nf=flag;
                if(!flag&&i<lmt){
                        nf=1;
                }
                ans+=get(pos+1,sum+i,nf);
        }
        return (dp[pos][sum][flag]=ans);
}
```

5.3 DP DNC

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ll dp[809][8009],ind[809][8009],c[8009],a[8009];
ll cost(int i,int j){
        if(i>j)return 0;
        ll sum=(c[j]-c[i-1])*(j-i+1);
        return sum;
}
void go(int g,int l,int r,int start_ind,int end_ind){
        if(l>r)return ;
        int mid=(l+r)/2;
        dp[g][mid]=LLONG_MAX;
        for(int i=start_ind;i<=end_ind;i++){
                ll cur=dp[g-1][i]+cost(i+1,mid);
                if(cur<dp[g][mid]){
                        dp[g][mid]=cur;
                        ind[g][mid]=i;
                }
        }
        go(g,l,mid-1,start_ind,ind[g][mid]);
        go(g,mid+1,r,ind[g][mid],end_ind);
}
int main(){
        int n,G;cin>>n>>G;
        for(int i=1;i<=n;i++){
                cin>>a[i];
                c[i]=a[i]+c[i-1];
        }
        for(ll i=1;i<=n;i++){
                dp[1][i]=c[i]*i;
        }
        for(int i=2;i<=G;i++){
                go(i,0,n,0,n);
```

```
        }
        cout<<dp[G][n];
}
```

5.4 Euclidean

```
ll mod(ll a, ll b)
// return a % b (positive value)
{
    while(a<0)a += b;
    return (a%b); }
ll gcd(ll a, ll b) {ll r; while (b)
    {r = a % b; a = b; b = r;} return a;} // computes
        gcd(a,b)
ll lcm(ll a, ll b) {return a / gcd(a, b) * b;} //
    computes lcm(a,b)
// returns d = gcd(a,b); finds x,y such that d = ax + by
ll extended_euclid(ll a, ll b, ll x, ll y) {
    ll xx = y = 0;ll yy = x = 1;
    while (b) {
        ll q = a/b,t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}
// finds all solutions to ax = b (mod n)
vector<ll> modular_linear_equation_solver(ll a, ll b, ll
     n) {
    ll x,y;
    vector<ll>solutions;
    ll d = extended_euclid(a, n, x, y);
    if (!(b%d)) {
        x = mod (x*(b/d), n);
        for (ll i = 0; i < d; i++)
            solutions.push_back(mod(x + i*(n/d),n));
    }
    return solutions;
}
// computes x and y such that ax + by = c; on failure, x
     = y =-1

// Note that solution exists iff c is a mulltiple of gcd
    (a,b)
void linear_diophantine(ll a, ll b, ll c, ll &x, ll &y)
    {
    ll d = gcd(a,b);
    if (c%d)
        x = y = -1;
    else {
        extended_euclid(a,b,x,y);
        x = x*(c/d);y = y*(c/d);
    }
}
// Function to find modulo inverse of a number in log(m)
ll modInverse(ll a, ll m){
    ll x, y;
    ll g = extended_euclid(a, m, x, y);
    if (g != 1) return -1; // Inverse mod d o e s n t
        exist
    ll res = (x%m + m) % m;
    return res;
}
```

5.5 Factors in n-1-3

```
//divisors in cube root n, pr is sieve
inline ll randll(){
  return  ( (ll)rand() << 30 ) + ( rand() << 15 ) + rand
      ();
}
inline ll mult(ll a, ll b, ll n){
  ll res = 0ll;
  a %= n, b %= n;
  while(b)
  {
    if(b&1) res = ( res + a ) % n;
    a = ( a + a ) % n;
    b >>= 1ll;
  }
  return res;
}
long long power(long long x,long long p,long long mod){
    long long s=1,m=x;
    while(p) {
        if(p&1) s=mult(s,m,mod);
        p>>=1;
        m=mult(m,m,mod);
    }
    return s;
}
bool witness(long long a,long long n,long long u,int t){
    long long x=power(a,u,n);
    for(int i=0;i<t;i++) {
        long long nx=mult(x,x,n);
        if(nx==1&&x!=1&&x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
```

```
bool millerRabin(long long n,int s=100) {
    if(n<2) return 0;
    if(!(n&1)) return n==2;
    long long u=n-1;
    int t=0;
    while(u&1) {
        u>>=1;
        t++;
    }
    while(s--) {
        long long a=randll()%(n-1)+1;
        if(witness(a,n,u,t)) return 0;
    }
    return 1;
}
inline bool isPr(ll n){
  return millerRabin( n , 1000 );
}
#define K 1000010
ll ans=1;
ll count_div_in_cube_root_n(ll n){
  for( ll i=2;i<K&&i<=n;i++)if(!pr[ i ])
    if(n%i==0){
        ll tcnt = 0;
        while( n % i == 0 )
                tcnt++,n/=i;
        ans*=(tcnt+1ll);
        }
        if(n!=1){
        ll tmp=sqrt( n );
        if( isPr( n ) ) ans*=2ll;
        else if( tmp * tmp == n ) ans*=3ll;
        else ans*=4ll;
        }
        return ans;
}
```

5.6 Fibo logn

```
ll fib(ll n,ll mod){
        ll i,h,j,k,t;i=h=1;j=k=0;
        while(n>0) {
                if(n%2==1)
                t=(j*h)%mod,j=(i*h + j*k +t)%mod,i=(i*k
                    + t)%mod;
                t=(h*h)%mod; h=(2*k*h + t)%mod;
                k=(k*k + t)%mod;n= n/2;
        }
        return j;
}
```

5.7 EGaussian Algorithm

```
//Gaussian elimination
const double EPS = 1e-9;
vector<double> GaussianElimination(const vector<vector<
    double> >& A, const vector<double>& b) {
    int i, j, k, pivot, n = A.size();
    vector<vector<double> > B(n, vector<double>(n+1));
    vector<double> x(n);
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) B[i][j] = A[i][j];
        B[i][n] = b[i];
    }
    for(i = 0; i < n; i++) {
        for(pivot = j = i; j < n; ++j) if(fabs(B[j][i])
            > fabs(B[pivot][i])) pivot = j;
        swap(B[i], B[pivot]);
        if(fabs(B[i][i]) < EPS) return vector<double>();
        for(j = n; j >= i; --j) B[i][j] /= B[i][i];
        for(j = 0; j < n; j++) if(i != j) for(k = i+1; k
            <= n; ++k) B[j][k] -= B[j][i] * B[i][k];
    }
    for(i = 0; i < n; i++) x[i] = B[i][n];
    return x;
}
```

5.8 Lucas theorem

```
//lucas thm
ll fact[14258+2];
ll ncr(ll n,ll r, ll MOD){
        if(r>n)return 0;
        ll num=fact[n]%MOD;
        ll den=fact[r]%MOD*fact[n-r]%MOD;
        den=den%MOD;
        return (num*inv(den,MOD))%MOD;
}
ll lucas(ll n, ll r,ll MOD){
        if(r>n)return 0;
        /*
        precompute in main
        ms(fact,0,sz fact);
        fact[0]=fact[1]=1;
        for(int i=2;i<=MOD;i++){
                fact[i]=i*fact[i-1];
                fact[i]%=MOD;
        }*/
        vector<ll> nn,rr;
```

```
        ll tn=n,tr=r,rem=0;
        while(tn){
                rem=tn%MOD;
                nn.pb(rem);
                tn=tn/MOD;
        }
        rem=0;
        while(tr){
                rem=tr%MOD;
                rr.pb(rem);
                tr=tr/MOD;
        }
        ll ans=1;
        for(int i=0;i<rr.size();i++){
                ans=ans*ncr(nn[i],rr[i],MOD)%MOD;
                ans=ans%MOD;
        }
        return ans;
}
```

5.9 Matrix expo

```
// rec relation: Ai=c1*Ai-1+c2*Ai-2+...ck*Ai-k
//A0=a0 A1=a1 ... Ak-1=ak-1
void multiply(ll F[2][2], ll M[2][2]);
void power(ll F[2][2], ll n);
ll ini[2];
ll fib(ll n){
  ll F[2][2] = {{0,-1},{1,(2*f)%MOD}};
  // F= (0 0 0 ..   ck)
  //    (1 0 0 ...ck-1)
  //      (0 1 0 ...ck-2)
  //    (.............)
  //      (0 0 0 ..1  c1)
  if (n == 1)return (ini[1]*I)%MOD;     //ini is [a0,a1
      ...,ak]
  power(F,n-1);
                //n-1 => n-k+1
  ll ans=(ini[1]%MOD*F[1][1]%MOD)%MOD+(ini[0]%MOD*F
      [0][1]%MOD)%MOD;
  if(ans<0)ans=(ans+MOD)%MOD;
  ans=(ans*I)%MOD;
  return ans;
}
void power(ll F[2][2], ll n){
  if( n == 0 || n == 1)
      return;
  ll M[2][2] = {{0,-1},{1,(2*f)%MOD}};
  power(F, n/2);
  multiply(F, F);
  if (n%2 != 0) multiply(F, M);
}
void multiply(ll F[2][2], ll M[2][2]){
  ll x =  (F[0][0]%MOD*M[0][0]%MOD + F[0][1]%MOD*M
      [1][0]%MOD)%MOD;
  ll y =  (F[0][0]%MOD*M[0][1]%MOD + F[0][1]%MOD*M
      [1][1]%MOD)%MOD;
  ll z =  (F[1][0]%MOD*M[0][0]%MOD + F[1][1]%MOD*M
      [1][0]%MOD)%MOD;
  ll w =  (F[1][0]%MOD*M[0][1]%MOD + F[1][1]%MOD*M
      [1][1]%MOD)%MOD;
  if(x<0)x=(x+MOD)%MOD;
  if(y<0)y=(y+MOD)%MOD;
  if(z<0)z=(z+MOD)%MOD;
  if(w<0)w=(w+MOD)%MOD;
  F[0][0] = x;
  F[0][1] = y;
  F[1][0] = z;
  F[1][1] = w;
}
```

5.10 Miller-Rabin

```
bool miller_rabin_primality(ll N){
        static const int p
           [12]={2,3,5,7,11,13,17,19,23,29,31,37};
        if(N<=1)return false;
        for(int i=0;i<12;++i){
                if(p[i]==N)return true;
                if(N%p[i]==0)return false;
        }
        ll c =N-1,g=0;
        while(!(c&1))c>>=1,++g;
        for(int i=0;i<12;++i){
                ll k=fpow(p[i],c,N);
                for(int j=0;j<g;++j){
                        ll kk=mult(k,k,N);
                        if(kk==1&&k!=1&&k!=N-1)
                                return false;
                        k=kk;
                }
                if(k!=1)
                        return false;
        }
        return true;
}
```

5.11 Mobius

```
//mobius
int mobius(ll n){
	prime.clear(); //primes till n
	pf(n);
	int c[1000000]={0};
	for(int i=0;i<prime.size();i++){
		c[prime[i]]++;
	}
	for(int i=1;i<1000000;i++){
		if(c[i]>=2)return 0;
	}
	if(prime.size()&1)return -1;
	return 1;
}
```

5.12 SQRT CBRT tourist

```
ll my_sqrt(ll x) {
    assert(x > 0);
    ll y = (ll) (sqrtl((ld) x) + 0.5);
    while (y * y < x)
        y++;
    while (y * y > x)
        y--;
    if (y * y == x)
        return y;
    return -1;
}
ll my_cbrt(ll x) {
    assert(x > 0);
    ll y = (ll) (powl((ld) x, 1.0 / 3.0) + 0.5);
    while (y * y * y < x)
        y++;
    while (y * y * y > x)
        y--;
    if (y * y * y == x)
        return y;
    return -1;
}
```

5.13 Euler totient

```
//phi
int totient[100008];
void phi(){
	for(int i=1;i<=100000;i++){
		int ans=i;
		set<int> s;
		int temp=i;
		while(temp!=1){
			s.insert(pr[temp]); //pr is spf
			temp/=pr[temp];
		}
		for(auto e:s){
			ans-=ans/e;
		}
		totient[i]=ans;
	}
}
```

5.14 FFT

```
const double PI = 4*atan(1);
const int N=2e5+5;
const int MOD=13313;

int FFT_N=0;
vector<base> omega;

void init_fft(int n)
{
	FFT_N = n;
	omega.resize(n);
	double angle = 2*PI/n;
	for(int i=0;i<n;i++)
	{
		omega[i]=base(cos(i*angle), sin(i*angle)
		    );
	}
}

void fft(vector<base> &a)
{
	int n=a.size();
	if(n==1)
		return;
	int half=n>>1;
	vector<base> even(half), odd(half);
	for(int i=0, j=0; i<n; i+=2, j++)
	{
		even[j]=a[i];
		odd[j]=a[i+1];
	}
	fft(even);
	fft(odd);
	int denominator=FFT_N/n;
	for(int i=0;i<half;i++)
	{
```

```
                base cur=odd[i] * omega[i*denominator];
                a[i]=even[i] + cur;
                a[i+half]=even[i] - cur;
        }
}

void multiply(vector<int> &a, vector<int> &b, vector<int
    > &res)
{
        vector<base> fa(a.begin(), a.end());
        vector<base> fb(b.begin(), b.end());
        int n=1;
        while(n<2*max(a.size(), b.size()))
                n<<=1;
        fa.resize(n);
        fb.resize(n);
        init_fft(n);
        fft(fa);
        fft(fb);
        for(int i=0;i<n;i++)
                fa[i] = conj(fa[i] *  fb[i]);
        fft(fa);
        res.resize(n);
        for(int i=0;i<n;i++)
        {
                res[i]=(long long)(fa[i].real()/n + 0.5)
                    ;
                res[i]%=MOD;
        }
}
```

5.15 FFT-DNC

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

#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.
    tie(0);
#define endl "\n"
#define int long long

typedef complex<double> base;

const double PI = 4*atan(1);
const int N=2e5+5;
const int MOD=13313;

int FFT_N=0;
vector<base> omega;

void init_fft(int n)
{
        FFT_N = n;
        omega.resize(n);
        double angle = 2*PI/n;
        for(int i=0;i<n;i++)
        {
                omega[i]=base(cos(i*angle), sin(i*angle)
                    );
        }
}

void fft(vector<base> &a)
{
        int n=a.size();
        if(n==1)
                return;
        int half=n>>1;
        vector<base> even(half), odd(half);
        for(int i=0, j=0; i<n; i+=2, j++)
        {
                even[j]=a[i];
                odd[j]=a[i+1];
        }
        fft(even);
        fft(odd);
        int denominator=FFT_N/n;
        for(int i=0;i<half;i++)
        {
                base cur=odd[i] * omega[i*denominator];
                a[i]=even[i] + cur;
                a[i+half]=even[i] - cur;
        }
}

void multiply(vector<int> &a, vector<int> &b, vector<int
    > &res)
{
        vector<base> fa(a.begin(), a.end());
        vector<base> fb(b.begin(), b.end());
        int n=1;
        while(n<2*max(a.size(), b.size()))
                n<<=1;
        fa.resize(n);
        fb.resize(n);
        init_fft(n);
        fft(fa);
        fft(fb);
        for(int i=0;i<n;i++)
                fa[i] = conj(fa[i] *  fb[i]);
```

```
        fft(fa);
        res.resize(n);
        for(int i=0;i<n;i++)
        {
                res[i]=(long long)(fa[i].real()/n + 0.5)
                    ;
                res[i]%=MOD;
        }
}

int n, k, q, curlen, idx=0;
int a[N], f[N];
vector<int> res;
vector<vector<int> > ans[40];

vector<int> divide(int lo, int hi)
{
        vector<int> ret;
        if(lo==hi)
        {
                ret.resize(f[lo]+1);
                for(int i=0;i<=f[lo];i++)
                        ret[i]=1;
                return ret;
        }
        int mid=(lo+hi)>>1;
        vector<int> v1=divide(lo, mid);
        vector<int> v2=divide(mid+1, hi);
        multiply(v1, v2, ret);
        ret.resize((int)v1.size()+(int)v2.size()-1);
        return ret;
}

int32_t main()
{
        IOS;
        cin>>n>>k;
        for(int i=1;i<=n;i++)
        {
                cin>>a[i];
                f[a[i]]++;
        }
        vector<int> ans=divide(1, 2e5);
        cout<<ans[k]<<endl;
        return 0;
}
```

6 Strings

6.1 Knuth-Morris-Pratt Algorithm

```
void compute(string pat, int lps[]){
        int len=0,m=pat.length();
        lps[0]=0;
        int i=1;
        while(i<m){
                if(pat[i]==pat[len]){
                        len++;
                        lps[i]=len;
                        i++;
                }else{
                        if(len!=0){
                                len=lps[len-1];
                        }else{
                                lps[i]=0;
                                i++;
                        }
                }
        }
}

void kmp(string text, string pat, int lps[]){
        compute(pat,lps);
        int i=0,j=0;
        while(i<text.length()){
                if(pat[j]==text[i]){
                        i++;
                        j++;
                }
                if(j==pat.length()){
                        cout<<"Found at "<<i-j<<"\n";
                        j=lps[j-1];
                }else if(pat[j]!=text[i] && i<text.
                    length()){
                        if(j!=0){
                                j=lps[j-1];
                        }else{
                                i++;
                        }
                }
        }
}

int main(){
        int lps[100];
```

```
        string pat,text;
        cin>>text>>pat;
        kmp(text,pat,lps);
        for(int i=0;i<pat.length();i++){
                cout<<lps[i];
        }
}
```

6.2 Suffix array

```
struct suffix{
    int index;
    int rank[2];
};
int cmp(struct suffix a, struct suffix b){
    return (a.rank[0] == b.rank[0])? (a.rank[1] < b.rank
        [1] ?1: 0):(a.rank[0] < b.rank[0] ?1: 0);
}
vector<int> buildSuffixArray(string txt){
    int n=txt.length();
    struct suffix suffixes[n];
    for (int i = 0; i < n; i++){
        suffixes[i].index = i;
        suffixes[i].rank[0] = txt[i] - 'a';
        suffixes[i].rank[1] = ((i+1) < n)? (txt[i + 1] -
            'a'): -1;
    }
    sort(suffixes, suffixes+n, cmp);
    int ind[n];
    for (int k = 4; k < 2*n; k = k*2){
        int rank = 0;
        int prev_rank = suffixes[0].rank[0];
        suffixes[0].rank[0] = rank;
        ind[suffixes[0].index] = 0;
        for (int i = 1; i < n; i++){
            if (suffixes[i].rank[0] == prev_rank &&
                suffixes[i].rank[1] == suffixes[i-1].
                rank[1]){
                 prev_rank = suffixes[i].rank[0];
                 suffixes[i].rank[0] = rank;
            }
            else{
                 prev_rank = suffixes[i].rank[0];
                 suffixes[i].rank[0] = ++rank;
            }
            ind[suffixes[i].index] = i;
        }
        for (int i = 0; i < n; i++){
            int nextindex = suffixes[i].index + k/2;
            suffixes[i].rank[1] = (nextindex < n)?
                suffixes[ind[nextindex]].rank[0]: -1;
        }
        sort(suffixes, suffixes+n, cmp);
    }
    vector<int>suffixArr(n);
    for (int i = 0; i < n; i++)
        suffixArr[i] = suffixes[i].index;
    return suffixArr;
}
```

6.3 z-function

```
//z-function
vector<ll>z(100001,0);
void calculatez(string &s){ // z[i] is the length of the
      longest substring starting from s[i] which is also
     a prefix of s
    ll n=s.size();
    z[0]=n;
    for(ll 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;
        }
    }
}
```

7 EZPZ

7.1 Template

```
#include<bits/stdc++.h>
#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define MOD 1000000007
#define MOD9 1000000009
#define pi 3.1415926535
#define ms(s, n) memset(s, n, sizeof(s))
#define prec(n) fixed<<setprecision(n)
#define eps 0.000001
#define all(v) v.begin(), v.end()
```

```
#define allr(v) v.rbegin(), v.rend()
#define bolt ios::sync_with_stdio(0)
#define light cin.tie(0);cout.tie(0)
#define forr(i,p,n) for(ll i=p;i<n;i++)
#define MAXN 1000003
typedef int ll;
using namespace std;
ll mult(ll a,ll b, ll p=MOD){return ((a%p)*(b%p))%p;}
ll add(ll a, ll b, ll p=MOD){return (a%p + b%p)%p;}
ll fpow(ll n, ll k, ll p = MOD) {ll r = 1; for (; k; k
    >>= 1) {if (k & 1) r = r * n%p; n = n * n%p;} return
     r;}
ll inv(ll a, ll p = MOD) {return fpow(a, p - 2, p);}
ll inv_euclid(ll a, ll m = MOD){ll m0 = m;ll y = 0, x =
    1;if (m == 1)return 0;while (a > 1){ll q = a / m;ll
    t = m;m = a % m, a = t;t = y;y = x - q * y;x = t;}if
     (x < 0)x += m0;return x;}
//https://www.youtube.com/watch?v=40TRPnvs4JE
```

7.2 fast io

```
void scanint(int &x)
{
    register int c = gc();
    x = 0;
    int neg = 0;
    for(;((c<48 || c>57) && c != '-');c = gc());
    if(c=='-') {neg=1;c=gc();}
    for(;c>47 && c<58;c = gc()) {x = (x<<1) + (x<<3) + c
        - 48;}
    if(neg) x=-x;
}
```

7.3 LIS nlogn

```
//lis NLOGN
int lis(int a[],int n){
        ll dp[n+3];
        //int lis[n+3];
        //ms(lis,0,sz lis);
        dp[0]=-LLONG_MAX;
        for(int i=1;i<=n;i++){
                dp[i]=LLONG_MAX;
        }
        int anss=-1;
        for(int i=1;i<=n;i++){
                int l=1,r=n,ans;
                while(l<=r){
                        int mid=(l+r)/2;
                        if(a[i]<=dp[mid]){
                                ans=mid;
                                r=mid-1;
                        }else{
                                l=mid+1;
                        }
                }
                dp[ans]=a[i];
//              lis[i]=max(lis[i],ans);
                anss=max(anss,ans);
        }
        return anss;
}
```

7.4 MOs

```
int N, Q;

// Variables, that hold current "state" of computation
long long current_answer;
long long cnt[100];

// Array to store answers (because the order we achieve
    them is messed up)
long long answers[100500];
int BLOCK_SIZE;
int arr[100500];

// We will represent each query as three numbers: L, R,
    idx. Idx is
// the position (in original order) of this query.
pair< pair<int, int>, int> queries[100500];


// Essential part of Mo's algorithm: comparator, which
    we will
// use with std::sort. It is a function, which must
    return True
// if query x must come earlier than query y, and False
    otherwise.
inline bool mo_cmp(const pair< pair<int, int>, int> &x,
        const pair< pair<int, int>, int> &y)
{
    int block_x = x.first.first / BLOCK_SIZE;
    int block_y = y.first.first / BLOCK_SIZE;
    if(block_x != block_y)
        return block_x < block_y;
    return x.first.second < y.first.second;
}
```

```cpp
// When adding a number, we first nullify it's effect on current
// answer, then update cnt array, then account for it's effect again.
inline void add(int x)
{
    current_answer -= cnt[x] * cnt[x] * x;
    cnt[x]++;
    current_answer += cnt[x] * cnt[x] * x;
}

// Removing is much like adding.
inline void remove(int x)
{
    current_answer -= cnt[x] * cnt[x] * x;
    cnt[x]--;
    current_answer += cnt[x] * cnt[x] * x;
}

int main()
{
    cin.sync_with_stdio(false);
    cin >> N >> Q;
    BLOCK_SIZE = static_cast<int>(sqrt(N));

    // Read input array
    for(int i = 0; i < N; i++)
        cin >> arr[i];

    // Read input queries, which are 0-indexed. Store each query's
    // original position. We will use it when printing answer.
    for(int i = 0; i < Q; i++) {
        cin >> queries[i].first.first >> queries[i].first.second;
        queries[i].second = i;
    }

    // Sort queries using Mo's special comparator we defined.
    sort(queries, queries + Q, mo_cmp);

    // Set up current segment [mo_left, mo_right].
    int mo_left = 0, mo_right = -1;

    for(int i = 0; i < Q; i++) {
        // [left, right] is what query we must answer now.
        int left = queries[i].first.first;
        int right = queries[i].first.second;

        // Usual part of applying Mo's algorithm: moving mo_left
        // and mo_right.
        while(mo_right < right) {
            mo_right++;
            add(arr[mo_right]);
        }
        while(mo_right > right) {
            remove(arr[mo_right]);
            mo_right--;
        }

        while(mo_left < left) {
            remove(arr[mo_left]);
            mo_left++;
        }
        while(mo_left > left) {
            mo_left--;
            add(arr[mo_left]);
        }

        // Store the answer into required position.
        answers[queries[i].second] = current_answer;
    }

    // We output answers *after* we process all queries.
    for(int i = 0; i < Q; i++)
        cout << answers[i] << "\n";
    return 0;
}
```

$f(n) = O(g(n))$	iff $\exists$ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n) = \Omega(g(n))$	iff $\exists$ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
$f(n) = o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim\limits_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup\limits_{n\to\infty} a_n$	$\lim\limits_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
$\genfrac{[}{]}{0pt}{}{n}{k}$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
$\genfrac{\{}{\}}{0pt}{}{n}{k}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\genfrac{\langle}{\rangle}{0pt}{}{n}{k}$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\ldots\pi_n$ on $\{1, 2, \ldots, n\}$ with k ascents.
$\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \qquad \sum_{i=1}^{n}\binom{i}{m}H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\genfrac{\{}{\}}{0pt}{}{n}{1} = \genfrac{\{}{\}}{0pt}{}{n}{n} = 1$,

12. $\genfrac{\{}{\}}{0pt}{}{n}{2} = 2^{n-1} - 1$, **13.** $\genfrac{\{}{\}}{0pt}{}{n}{k} = k\genfrac{\{}{\}}{0pt}{}{n-1}{k} + \genfrac{\{}{\}}{0pt}{}{n-1}{k-1}$,

14. $\genfrac{[}{]}{0pt}{}{n}{1} = (n-1)!$, **15.** $\genfrac{[}{]}{0pt}{}{n}{2} = (n-1)!H_{n-1}$, **16.** $\genfrac{[}{]}{0pt}{}{n}{n} = 1$, **17.** $\genfrac{[}{]}{0pt}{}{n}{k} \ge \genfrac{\{}{\}}{0pt}{}{n}{k}$,

18. $\genfrac{[}{]}{0pt}{}{n}{k} = (n-1)\genfrac{[}{]}{0pt}{}{n-1}{k} + \genfrac{[}{]}{0pt}{}{n-1}{k-1}$, **19.** $\genfrac{\{}{\}}{0pt}{}{n}{n-1} = \genfrac{[}{]}{0pt}{}{n}{n-1} = \binom{n}{2}$, **20.** $\sum_{k=0}^{n}\genfrac{[}{]}{0pt}{}{n}{k} = n!$, **21.** $C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\genfrac{\langle}{\rangle}{0pt}{}{n}{0} = \genfrac{\langle}{\rangle}{0pt}{}{n}{n-1} = 1$, **23.** $\genfrac{\langle}{\rangle}{0pt}{}{n}{k} = \genfrac{\langle}{\rangle}{0pt}{}{n}{n-1-k}$, **24.** $\genfrac{\langle}{\rangle}{0pt}{}{n}{k} = (k+1)\genfrac{\langle}{\rangle}{0pt}{}{n-1}{k} + (n-k)\genfrac{\langle}{\rangle}{0pt}{}{n-1}{k-1}$,

25. $\genfrac{\langle}{\rangle}{0pt}{}{0}{k} = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\genfrac{\langle}{\rangle}{0pt}{}{n}{1} = 2^n - n - 1$, **27.** $\genfrac{\langle}{\rangle}{0pt}{}{n}{2} = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $x^n = \sum_{k=0}^{n}\genfrac{\langle}{\rangle}{0pt}{}{n}{k}\binom{x+k}{n}$, **29.** $\genfrac{\langle}{\rangle}{0pt}{}{n}{m} = \sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $m!\genfrac{\{}{\}}{0pt}{}{n}{m} = \sum_{k=0}^{n}\genfrac{\langle}{\rangle}{0pt}{}{n}{k}\binom{k}{n-m}$,

31. $\genfrac{\langle}{\rangle}{0pt}{}{n}{m} = \sum_{k=0}^{n}\genfrac{\{}{\}}{0pt}{}{n}{k}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\left\langle\!\!\left\langle {n \atop 0} \right\rangle\!\!\right\rangle = 1$, **33.** $\left\langle\!\!\left\langle {n \atop n} \right\rangle\!\!\right\rangle = 0$ for $n \ne 0$,

34. $\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle {n-1 \atop k-1} \right\rangle\!\!\right\rangle$, **35.** $\sum_{k=0}^{n}\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. $\genfrac{\{}{\}}{0pt}{}{x}{x-n} = \sum_{k=0}^{n}\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle\binom{x+n-1-k}{2n}$, **37.** $\genfrac{\{}{\}}{0pt}{}{n+1}{m+1} = \sum_{k}\binom{n}{k}\genfrac{\{}{\}}{0pt}{}{k}{m} = \sum_{k=0}^{n}\genfrac{\{}{\}}{0pt}{}{k}{m}(m+1)^{n-k}$, The Chinese remainder theorem: There exists a number C such that:

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^{n} p_i^{e_i - 1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^{n} p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^{n} \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$$

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \ldots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \lvert S \rvert$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k,\ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

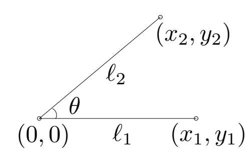
Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p},$$
$$\lim_{p \to \infty} \left[|x_1 - x_0|^p + |y_1 - y_0|^p\right]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\frac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos\theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3}\pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \cdots &&= \sum_{i=0}^{\infty} x^i,\\
\frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \cdots &&= \sum_{i=0}^{\infty} c^i x^i,\\
\frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \cdots &&= \sum_{i=0}^{\infty} x^{ni},\\
\frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \cdots &&= \sum_{i=0}^{\infty} ix^i,\\
x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots &&= \sum_{i=0}^{\infty} i^n x^i,\\
e^x &= 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{x^i}{i!},\\
\ln(1+x) &= x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots &&= \sum_{i=1}^{\infty} (-1)^{i+1}\frac{x^i}{i},\\
\ln\frac{1}{1-x} &= x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots &&= \sum_{i=1}^{\infty} \frac{x^i}{i},\\
\sin x &= x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},\\
\cos x &= 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},\\
\tan^{-1} x &= x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},\\
(1+x)^n &= 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{n}{i} x^i,\\
\frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,\\
\frac{x}{e^x - 1} &= 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots &&= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},\\
\frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,\\
\frac{1}{\sqrt{1-4x}} &= 1 + x + 2x^2 + 6x^3 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i}{i} x^i,\\
\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n &= 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,\\
\frac{1}{1-x}\ln\frac{1}{1-x} &= x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots &&= \sum_{i=1}^{\infty} H_i x^i,\\
\frac{1}{2}\left(\ln\frac{1}{1-x}\right)^2 &= \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots &&= \sum_{i=2}^{\infty} \frac{H_{i-1}x^i}{i},\\
\frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \cdots &&= \sum_{i=0}^{\infty} F_i x^i,\\
\frac{F_n x}{1-(F_{n-1}+F_{n+1})x-(-1)^n x^2} &= F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots &&= \sum_{i=0}^{\infty} F_{ni} x^i.
\end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\begin{aligned}
\alpha A(x) + \beta B(x) &= \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,\\
x^k A(x) &= \sum_{i=k}^{\infty} a_{i-k} x^i,\\
\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} &= \sum_{i=0}^{\infty} a_{i+k} x^i,\\
A(cx) &= \sum_{i=0}^{\infty} c^i a_i x^i,\\
A'(x) &= \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,\\
xA'(x) &= \sum_{i=1}^{\infty} i a_i x^i,\\
\int A(x)\,dx &= \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,\\
\frac{A(x) + A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i} x^{2i},\\
\frac{A(x) - A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.
\end{aligned}$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.
– Leopold Kronecker